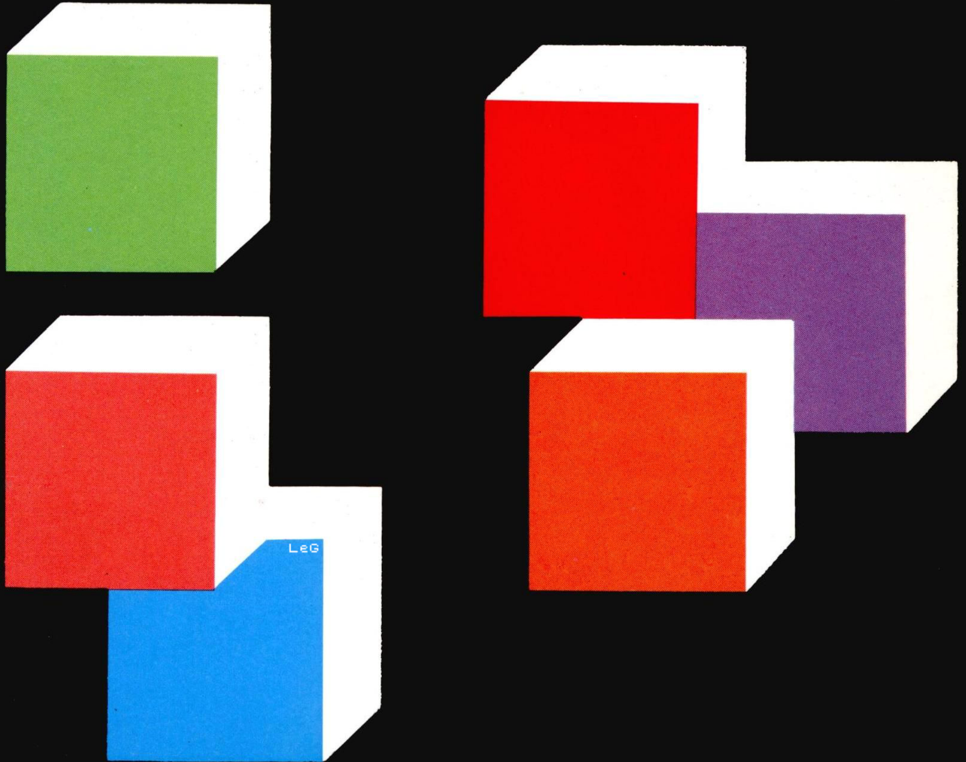
BASIC
curso acelerado



Claude. J. DeRossi

PARANINFO S A

BASIC
Curso acelerado

CLAUDE J. DE ROSSI

BASIC
Curso acelerado

QUINTA EDICION

1985

PARANINFO S.A.

MADRID

Traducido por:
GONZALO GARCIA LOPEZ DE HEREDIA

Título original inglés: LEARNING BASIC FAST
publicado por: RESTON PUBLISHING COMPANY, INC.
A Prentice-Hall Company, Reston, Virginia.

IMPRESO EN ESPAÑA
PRINTED IN SPAIN

ISBN: 0-8359-3977-4 (edición inglesa)
ISBN: 84-283-1289-3 (edición española)

Depósito Legal: M-35044-1985

PARANINFO S.A. Magallanes, 25 - 28015 MADRID (5-3615)

ALCO, artes gráficas. Jaspe, 34. 28026 MADRID

A Dan, Lorris, Mickey

Indice de materias

Prólogo

Este texto sobre el lenguaje de programación BASIC está dirigido a personas que deben aprender el lenguaje rápidamente. Supone que el lector no conoce nada sobre la materia y por consiguiente avanza desde los conceptos muy fundamentales hasta los del más sofisticado nivel.

Este libro está escrito en un estilo informal y no técnico. Como el texto se construye partiendo del conocimiento anterior, proporciona al estudiante la habilidad para practicar lo que conoce. Numerosos ejemplos y ejercicios refuerzan el conocimiento que el estudiante ha logrado.

Este libro puede usarse en un curso de un semestre sobre programación en BASIC o bien como un texto auto-suficiente para una persona que ha de aprender BASIC por sí misma.

Me complace expresar mi agradecimiento a las personas que me han ayudado en este esfuerzo. En particular doy las gracias a mi esposa, Cindy, por su experto mecanografiado.

Claude J. de Rossi

CAPITULO **1**

Introducción

El lenguaje de programación BASIC fue desarrollado en los años 60 para permitir que personas que no son programadores de ordenadores consigan la ayuda de un ordenador para obtener soluciones a problemas matemáticos y de gestión relacionados con los negocios. El lenguaje se desarrolló bajo la dirección de John G. Kemeny en el Dartmouth College. Desde sus comienzos en Darmouth, el lenguaje ha crecido notablemente en popularidad y se usa actualmente en muchas diferentes marcas de ordenadores.

Para usar el BASIC, una persona telefonea a un ordenador distante, situado de 8 metros a 8.000 km de donde el teléfono está localizado. Entonces, cuando el ordenador responde con un tono de alto timbre, el usuario coloca el receptor en una plataforma asociada a una máquina de escribir como terminal. El terminal da señales de vida y escribe un mensaje al usuario. El mensaje que aparece es como éste:

```
TIMESHARING SERVICE 4/9/83 14:28
ID–
```

El ordenador se ha identificado a sí mismo y a su vez ha indicado al usuario que haga lo mismo. Si el usuario ha hecho un contrato previo con el centro del ordenador, él puede escribir una ID válida, por ejemplo:

```
ID–HP638
```

El ordenador comprueba la ID, y si es válida, escribe

```
SYSTEM–
```

El usuario puede responder con ALGOL, FORTRAN, BASIC, PL/I o cualquier otro lenguaje de programación soportado por la instalación. El más fácil de aprender y de usar de estos lenguajes es BASIC. El usuario responde:

```
SYSTEM–BASIC
```

El ordenador hace una pregunta:

```
OLD OR NEW–
```

El usuario responde:

```
OLD OR NEW–NEW SIGMA
```

El usuario ha indicado que desea dar al ordenador un nuevo conjunto de instrucciones, el *programa,* para que él lo ejecute. Le da al programa el nombre de SIGMA. El usuario tenía la opción de escribir OLD si hubiera querido ejecutar un programa que él había previamente desarrollado y guardado en la instalación del ordenador. Porteriormente veremos como un usuario puede llamar a un programa viejo guardado antes.

El ordenador escribe:

READY (preparado)

El usuario es ahora libre de escribir varios comandos que constituyen un programa para que el ordenador lo ejecute. El siguiente es un ejemplo de programa que el usuario puede introducir:

```
10   DATA 5, 8, 31, 4, 16, 32, 999
20   LET S = 0
30   LET N = 0
40   READ X
50   IF X = 999 THEN 90
60   LET S = S + X
70   LET N = N + 1
80   GO TO 40
90   PRINT S/N
100  END
```

El usuario escribe estas líneas una cada vez. Al final de cada línea, pulsa una tecla llamada CARRIAGE RETURN (retorno de carro). Cuando pulsa la tecla CARRIAGE RETURN, el ordenador acepta el comando que el usuario ha terminado de escribir.

Vd. puede no entender el programa indicado anteriormente pero no se preocupe. Es sólo un ejemplo de un programa completo para que Vd. pueda ver cómo es un programa en BASIC. Posteriormente, se explicarán detalladamente las significaciones de los varios comandos utilizados. Para satisfacer al curioso, podría decirse que el programa calcula el promedio de los valores 5, 8, 31, 4, 16 y 32. A continuación imprime este valor medio.

Obsérvense los números enteros al comienzo de cada instrucción en BASIC. Estos números se llaman *números de línea.* Cada instrucción en BASIC debe tener un número de línea. Este número puede ser cualquiera que el usuario elija con tal que cada número de línea sea mayor que el de la línea que le precede.

Los números de línea de las diez instrucciones indicadas podían haber sido 1, 18, 35, 72, 146, 208, 231, 232, 306 y 543.

Cada comando en BASIC se llama una SENTENCIA.

El programa ejemplo tiene diez sentencias. Obsérvese que hay 10 líneas en el programa. Cada línea contiene una única sentencia.

Habiendo terminado el programa, el usuario puede querer ahora que el ordenador lo ejecute. Ejecutar un programa significa que proceda a analizar las instrucciones y luego seguirlas para obtener resultados.

El usuario escribe:

```
RUN
```

Obsérvese que el comando RUN no exige un número de línea. Es decir, no debe usarse ninguno.

El ordenador deberá ejecutar el programa y dar la respuesta. La respuesta es 16, de forma que el ordenador *escribe*:

```
16
```

El usuario *escribe:*

```
BYE
```

y la sesión de tiempo compartido ("time-sharing") ha terminado.

Para recapitular, veamos la conversación completa entre el ordenador y el usuario:

```
TIEMSHARING SERVICE 4/9/83 14:28
ID–HP638
SYSTEM–BASIC
OLD OR NEW–NEW
READY
10 DATA 5, 8, 31, 4, 16, 32, 999
20 LET S = 0
30    LET N = 0
40    READ X
```

```
50  IF X = 999 THEN 90
60  LET S = S + X
70  LET N = N + 1
80  GO TO 40
90  PRINT S/N
100 END
    RUN
    16
    BYE
                        OFF 4/9/83 14:70
```

¿QUE HA APRENDIDO VD.?

En este capítulo, Vd. ha aprendido que puede llamar a un ordenador distante por teléfono y tener una conversación con él. Todo lo que Vd. precisa es un teléfono, un terminal y una ID de usuario (identificación de usuario).

Vd. *escribe* su programa dando sentencia tras sentencia hasta que su programa ha entrado completamente. Para finalizar una línea conteniendo una sentencia, Vd. pulsa la tecla CARRIAGE RETURN. Esta acción provoca que el ordenador acepte la última sentencia *escrita*. Cada sentencia deberá estar precedida por un número de línea. Los números de línea son arbitrarios, pero deben darse en sucesión creciente.

Algunas de las palabras inglesas que se usan en el lenguaje de programación BASIC son:

```
 DATA
 LET
 READ
 IF
 PRINT
*GO TO
 END
```

Hay solamente unas pocas más.

Los dos comandos RUN y BYE son palabras especiales. No son parte del lenguaje BASIC, y por ello no deben estar precedidas por números de línea. Se discutirán RUN, BYE y otras palabras cuando se estudien los *comandos del sistema* en el capítulo 10.

---

* GO TO se supone también que es una sola palabra y puede escribirse GOTO.

Ejercicios y Cuestions

1. ¿Cuál es el lenguaje de programación descrito en este libro?
2. ¿A qué distancia mínima y máxima del ordenador puede instalarse un terminal?
3. ¿Qué es una ID?
4. ¿De qué otros lenguajes de programación se disponen en la modalidad de tiempo compartido?
5. Cuando un usuario telefonee a un ordenador distante, ¿cómo sabe que el ordenador le ha respondido?
6. ¿Qué dos items de información debe dar un ordenador cuando responde a la llamada de un usuario?
7. ¿Qué desea conocer el ordenador cuando escrite OLD OR NEW?
8. Cuando una persona escribe NEW, ¿qué debe escribir a continuación de dicha palabra?
9. ¿Cuál es la definición de una sentencia en BASIC?
10. ¿Qué es un número de línea?
11. ¿Los números de línea deben aumentar siempre en saltos de 10?
12. Dar nueve palabras que puedan utilizarse cuando se esté usando el lenguaje BASIC en el modo de tiempo-compartido.
13. ¿Qué dos de las palabras anteriores se llaman comandos del sistema?
14. ¿Puede Vd. situar números de línea en la cabecera de los comandos del sistema?
15. ¿Qué tecla debe pulsar el usuario al final de cada línea?
16. ¿Qué significa la frase *ejecutar un programa*?
17. ¿Qué comando del sistema debe usar cuando quiere terminar una sesión de tiempo-compartido?

CAPITULO **2**

Uso eficiente del sistema

Existen unos pocos hechos que Vd. debe conocer concernientes a la forma apropiada de usar su terminal mientras está programando en BASIC. Suponemos que Vd. está ansioso de comenzar a aprender el lenguaje y a descubrir cómo usarlo en la resolución de problemas, pero por favor espere con nosotros un capítulo más.

Cuando Vd. marque el número telefónico del ordenador, Vd. escuchará un tono de alto timbre procedente del receptor. El tono le hace saber a Vd. que está conectado. Sitúa el receptor en la plataforma especial asociada a su terminal. Vd. está ahora usando un dispositivo llamado acoplador "acústico".

No todos los terminales son iguales. Vd. puede hallar que hay otra forma de conesión para usar su terminal particular. Si es así, simplemente pregunte a alguien que haya utilizado el terminal. Estas personas se complacerán en indicarle cómo establecer la conexión con el ordenador distante.

Mientras Vd. está conectado, está operando en un modo llamado tiempo-compartido conversacional. Este modo se llama "conversacional" porque Vd. y el ordenador son interactivos: Vd. *escribe* algo, y el ordenador responde. Entonces Vd. *escribe* algo más, y el ordenador replica. De esta forma Vd. y el ordenador pueden comunicarse uno con otro, conectando lo que Vd. desea hacer y como lo está haciendo.

Por ejemplo, si Vd. comete errores, el ordenador le hablará sobre ello y le dará una posibilidad de corregirlos.

El modo se llama "tiempo-compartido" (timesharing) porque Vd. comparte el tiempo del ordenador distante con muchos otros usuarios, algunas veces cientos de usuarios. Como un ordenador es muy rápido, y puesto que durante gran parte del tiempo en que él está conectado, cada uno de los usuarios no está pidiendo al ordenador muchas cosas por hacer, el ordenador tiene poca dificultad en obedecer rápidamente los comandos de sus muchos usuarios de tiempo compartido.

Cuando el sistema pide un número ID, Vd. debe darle una ID válida. Es fácil el contratar una ID. Será facturado una vez al mes por el servicio. Mensualmente los cargos están basados en el tiempo en el que Vd. está conectado actualmente al ordenador distante (aproximadamente $5.00 por hora), el tiempo actual de ordenador usado (aproximadamente $1.00 por segundo), y el coste de alquiler de un terminal (aproximadamente $150.00 por mes). Existe también una variedad de pequeños cargos que los usuarios pueden evitar si lo desean.

Habiéndole Vd. respondido con una identificación de usuario o ID válida, el ordenador pregunta qué lenguaje quiere usar. Están disponibles otros lenguajes además del BASIC, pero el BASIC es más fácil que todos los demás, de aprender y de usar.

El sistema pregunta OLD OR NEW. Si Vd. ha escrito un programa anteriormente y desea usarlo ahora, puede llamar al programa antiguo suponiendo que lo había guardado bajo algún nombre de fichero –por ejemplo AREACU o RAIZ. Más tarde le indicaremos cómo salvar programas. Si Vd. quiere construir un nuevo programa desde el comienzo, Vd. responde NEW y da el nombre del nuevo programa.

Cuando el sistema *escribe* READY (preparado), indica que:

1. Vd. puede comenzar a construir un nuevo programa; o que
2. puede iniciar el uso de un antiguo programa que el sistema ha recuperado para Vd.

Al entrar un programa nuevo, la persona escribe una serie de sentencias, cada una de ellas encabezada por un número de línea. Si la persona escribe las sentencias en una sucesión fortuita, el ordenador las colocará en orden con arreglo a sus números de línea (en sucesión creciente) cuando el usuario *escriba* RUN. En este momento, el ordenador antes de obedecer el comando RUN, comprobará la existencia de posibles errores en el programa correspondiente. Si el usuario entra dos sentencias con el mismo número de línea, el sistema acepta solamente la *última* sentencia escrita.

Más adelante en este libro, discutiremos comandos especiales llamados *comandos del sistema*. Vd. ha sido informado ya de dos comandos del sistema, RUN y BYE. Se discutirán comandos adicionales del sistema como OLD, NEW, LIST, SAVE y otros.

Nadie es perfecto. Cuando cometamos errores, los corregimos de una forma sencilla.

Veamos aquí cinco maneras de hacer correcciones:

1. Supongamos que Vd. entra una sentencia en BASIC e inmediatamente observa que es incorrecta. Vd. puede corregir la sentencia escribiéndola de nuevo. Observemos cómo se corrige la sentencia de la línea 10.

```
5   LET X = 25.6
10  LET P = 6.9.8
10  LET P = 69.8
20  LET R = 3.94
```

La segunda sentencia en la línea 10 sustituye a la primera.

Vd. puede corregir una sentencia cuando advierte que lo escrito era erróneo. Observemos cómo se corrige la sentencia de la línea 10:

```
10  PRINT "ESTO ES UNA XXXX"
20  PRINT 92 + 65
30  PRINT 86 / 67
40  PRINT 6 * 27
99  END
10  PRINT "ESTO ES UNA DEMO"
```

La segunda sentencia de la línea 10 sustituye a la primera.

2. Mientras escribe Vd. una línea puede advertir que ha confundido uno o más caracteres. Vd. puede cambiar aquellos caracteres antes de dar el retorno de carro. Para hacerlo, desplace el carro hacia atrás empleando el símbolo ←. (Pulsando la tecla con dicho símbolo ←). El sistema va hacia atrás una posición o carácter por cada ←. Observemos cómo se han hecho las correcciones en las sentencias 10, 20 y 30 a continuación:

```
10  LET W = 7.65 ← 4
20  LET X = 4.92 ← 37
30  LET Z = 7.63 ←←←← 8.63
```

El ordenador aceptará las tres sentencias así

```
10  LET W = 7.64
20  LET X = 4.937
30  LET Z = 8.63
```

Adviértase que en algunos sistemas BASIC, el carácter usado para volver atrás el carro no es ←, sino @ ó \.

3. Mientras se escribe una sentencia, Vd. puede advertir que preferiría no escribirla. En este caso pulsa las teclas CTRL y X simultáneamente. El sistema

responderá DEL y la sentencia que Vd. había comenzado a escribir será ignorada. Estudiar cómo la línea 100 es corregida a continuación:

```
100  LET F = 6.9
200  LET G = F + 3.6
300  LET S = G + F
200  LET F = 7 DEL
100  LET F = 7.9
```

El usuario ha intentado corregir la línea 100, pero ha comenzado corrigiendo en su lugar la línea 200. Cuando descubre este error (después de haber escrito 200 LET F = 7), pulsa las teclas CRTL y X simultáneamente. El sistema escribe DEL. Así la línea 200 no ha sido alterada por la acción del usuario.

4. Para borrar una sentencia ya tecleada, escribir únicamente su número de línea y entonces dar el retorno de carro.

Ejemplo:

```
10  LET P = 6
20  LET P = 8
30  LET Q = 7
40  PRINT P * Q
50  END
20
```

La línea 20 es borrada del programa.

5. Para insertar una sentencia, se elige un número de línea no usado entre los números de línea de las sentencias adyacentes.

```
10  LET W = 4
20  PRINT W + X
15  LET X = 2
30  END
```

La línea 15 se inserta entre las líneas 10 y 20.

A partir de ahora comenzaremos el estudio del lenguaje de programación BASIC.

EJERCICIOS Y CUESTIONES

1. ¿Cuál es el nombre del modo en el que Vd. emplea generalmente el lenguaje de programación BASIC?
2. ¿Por qué se usa la palabra *conversacional* en conexión con el término tiempo compartido?
3. Verdadero o falso. Si Vd. comete un error mientras esté operando en el modo de tiempo compartido no puede Vd. hacer nada hasta el día siguiente.
4. ¿Qué significa el término tiempo-compartido?
5. ¿En qué consiste la habilidad principal de un ordenador que le permite atender a muchos usuarios en tiempo compartido simultáneamente?
6. Decir las tres componentes que contribuyen a su cargo mensual por el servicio de tiempo compartido.
7. Aproximadamente ¿cuánto debe pagar por hora una persona simplemente por estar conectado a un ordenador a distancia?
8. Aproximadamnte ¿cuánto debe pagar una persona por segundo, por el tiempo actual de ordenador usado cuando le pide al odenador que ejecute un programa?
9. Aproximadamente ¿cuál es el coste del alquiler mensual de un terminal para ser usado en tiempo compartido?
10. Cuando el ordenador escribe READY (preparado) ¿qué dos posibilidades se le ofrecen a Vd.?
11. ¿Cuáles son el mayor y el menor número de línea que Vd. puede usar cuando elige un número de línea para cada sentencia en BASIC?
12. ¿Qué hará el ordenador con las sentencias que Vd. escriba si las entra fuera de secuencia?
13. Para retroceder el carro con la finalidad de hacer correcciones ¿qué carácter debe emplear?
14. Supongamos que Vd. ha escrito

    ```
    150  LET P = (X + Y/Z)
    ```

    pero había pretendido escribir

    ```
    150  LET P = (X + Y * Z)/Q
    ```

    ¿Qué caracteres debe escribir siguiendo a Z para entrar la sentencia correctamente?
15. Si antes de retroceder el carro, Vd. advierte que no desea completar una sentencia que había comenzado a escribir, ¿qué *dos* teclas debe pulsar que borren la entrada entera?

16. ¿Qué comando del sistema debe escribir cuando desee que el ordenador ejecute su programa?
17. ¿Cuáles son los seis comandos del sistema mencionados en este capítulo?
18. ¿Qué hará el ordenador antes de obedecer actualmente al comando del sistema RUN?
19. ¿Qué sentencia acepta el ordenador para un programa, si Vd. escribe dos sentencias que tienen el mismo número de líneas?
20. ¿Cómo debe Vd. dirigir al ordenador para borrar una sentencia no deseada que Vd. había entrado previamente en el programa?
21. ¿Cómo puede Vd. insertar una sentencia que había olvidado entrar cuando había escrito originariamente su programa?

CAPITULO **3**

La sentencia LET

Al aprender cómo usar el lenguaje de programación BASIC, Vd. únicamente ha de conocer unos pocos tipos de sentencias. Estos tipos comienzan con las palabras LET, IF, PRINT, READ, GO TO, FOR, NEXT, STOP y END, y únicamente existen unas pocas más.

Los diversos tipos de sentencia se identifican por la primera palabra de la sentencia. Así, podremos referirnos a la sentencia LET, a la sentencia IF, a la sentencia PRINT, etc.

En los capítulos siguientes, aprenderemos solamente unos pocos tipos de sentencias: las sentencias LET, IF, PRINT, GO TO y END. Vd. comprobará que con el conocimiento de únicamente cinco tipos de sentencias, es capaz de dirigir a un ordenador a resolver maravillosamente gran variedad de problemas prácticos.

Vamos a considerar en primer lugar la sentencia LET. La sentencia LET se usa para asignar un número dado o el resultado de un cálculo a un nombre.

Es la sentencia más sencilla. La forma general es:

LET < variable > = expresión

Se puede considerar como un comando que ejecuta dos acciones distintas:

1. Produce el valor de la expresión que se halla a la derecha del signo =.
2. Asigna este valor a la variable que se halla a la izquierda del signo =.

Por consiguiente LET V = A conviene leerla como:

el valor de V se convierte en A, ó V es sustituido por A

Veamos aquí unos ejemplos:

```
10 LET P = 3.4
20 LET S = 9.1
30 LET T = 3.1 + 5.8
40 LET V = P + 5
50 LET W = P – S
```

En el ejemplo anterior y en la línea 10, el valor 3.4 es asignado a una celda de la memoria del ordenador llamada P.

Cuando P aparece luego en las últimas sentencias (por ejemplo en las líneas 40 y 50), el valor de P en estas sentencias es aquél que le fue asignado previamente. En el ejemplo, el valor de P usado en las líneas 40 y 50 es 3.4.

El valor 9.1 se asigna entonces a S (ver línea 20).

En la línea 30, la suma de 3.1 y 5.8 (8.9) se la asigna a T.

En la línea 40 la suma de P (3.4) y 5 es asignada a V. Esta suma es 8.4. En la línea 50, el valor de S (9.1) es restado de P (3.4) y el resultado se asigna a W. Este resultado es –5.7.

Después de que las cinco sentencias LET han sido ejecutadas por el ordenador, los valores de P, S, T, V y W, tal y como ellos son almacenados en las cinco celdas de la memoria, son éstos:

3.4	9.1	8.9	8.4	–5.7
P	S	T	V	W

En BASIC, los nombres de las celdas de la memoria pueden estar formados por una única letra del alfabeto tal como A, B, C, P, Q, R, X, Y, Z ó bien por una letra única seguida por un único dígito.

Ejemplos:

A1, A2, B6, C0, E6, F9, G5

Los siguientes nombres para celdas de memoria en lenguaje BASIC son correctos:

D
A8
F
H6
M
N0

Son incorrectos:

5M (Los nombres no pueden comenzar con un dígito)
XD (Los nombres no pueden consistir en dos letras)
P6X (Los nombres no pueden contener más que dos caracteres)

Las sentencias LET pueden utilizar el mismo nombre a ambos lados del signo igual. Una sentencia como

```
50  LET D = D + 6
```

es correcta. El valor primitivo de D es sumado a 6, y la suma calculada se asigna de nuevo a la celda de memoria llamada D. Por consiguiente si D tenía el valor

8.94
D

antes de que la sentencia fuese ejecutada; luego valdría

14.94
D

después de que la sentencia fuese ejecutada.

Obsérvese lo siguiente: en cada sentencia LET usada para un cálculo, el cálculo entero se efectúa antes que se lleve a cabo la asignación final del resultado a una celda de la memoria.

Ejemplo:

```
60  LET F = G + 13.44 + F
```

Si suponemos que el valor de G era 9.2 y el de F era 2.2 antes del cálculo, el cálculo entero se lleva a cabo *antes* que la asignación final 60 sea ejecutada, F y G son:

2.2	9.2
F	G

Tras la ejecución de la sentencia, F y G quedan:

24.84	9.2
F	G

Estudiemos el siguiente segmento de programa:

```
10  LET M = 6
20  LET N6 = 17.3
30  LET P = M + N6
40  LET M = M + 10
50  LET W4 = N6 − M
```

Llenemos estas celdas con los valores resultantes, suponiendo que las sentencias han sido ejecutadas:

□	□	□	□
M	N6	P	W4

El valor correcto a situar en M es 16. El valor asignado originalmente a M era 6, pero posteriormente se le sumó 10 (véase las sentencias en las líneas 10 y 40).

El valor correcto a colocar en P es 23.3. Este valor se le asignó en la línea 30. En ese momento el valor de M era 6. Cuando el valor de M fue cambiado a 16 en la línea 40, el ordenador no retrocedió para cambiar el valor de P que le había asignado en la línea 30.

El valor correcto a situar en W4 es 1.3. El valor de W4 fue asignado en la línea 50. En ese momento el valor de N6 era 17.3 y el valor de M era 16.

En BASIC, los nombres a los que nunca se les han dado valores se supone que contienen ceros. Así el programa

```
10  LET A = 4
20  LET C = A + B
30  PRINT A, B, C
40  END
```

funcionará, pero ha sido malamente escrito. El programa imprimirá 4, 0 y 4.

EJERCICIOS Y CUESTIONES

1. ¿Cómo se identifica el tipo de sentencia en BASIC?
2. ¿Cuáles son los primeros cinco tipos de sentencias sobre las cuales se concentra este texto?
3. Verdadero o falso. Estos cinco tipos anteriores de sentencia le permiten programar las soluciones a una extensa variedad de problemas.
4. ¿Cuál es la función de la sentencia LET?
5. En conexión con la sentencia LET, ¿qué significa el término asignar?
6. ¿Cuáles son en BASIC, las reglas para construir nombres para las celdas de la memoria del computador?
7. Verdadero o falso. Estos nombres en BASIC para celdas de la memoria son todos correctos: X; P3; Q; W5.
8. Verdadero o falso. ¿Estos nombres en BASIC para celdas de memoria son todos correctos?: 3W; FL; A34.
9. Explicar lo que estaba mal en cada nombre incorrecto en BASIC que Vd. encontró en las cuestiones 7 y 8.
10. Explicar qués es lo que significa la sentencia de asignación:

```
200  LET T = 6.2
```

11. Explicar qué significa la sentencia de asignación:

```
200  LET T = 4.6 + T
```

12. Estudiar este programa y decir lo que el ordenador imprime cuando ejecuta la sentencia de la línea 250.

```
200  LET L = 4
210  LET P = 7.2
220  LET Q = P – 3.6
230  LET R = Q + L
240  LET P = P + 2.1
250  PRINT L, P, Q, R
260  LET L = 2
270  END
```

(Una sentencia PRINT imprime siempre el último valor que se le había asignado previamente a un nombre en BASIC.)

13. Encontrar cuatro errores en el programa siguiente:

```
300  LEM M = 4
310  LET W6 = 2.6
```

```
320  LET DTL = 7
330  LET N = 7.5.3
340  PRINT M, W6, N
350  DONE
```

14. Verdadero o falso. El programa escrito a continuación funcionará bien:

```
400  LET A = B + C
410  PRINT A, B, C
420  END
```

CAPITULO **4**

La sentencia IF

Usando la sentencia IF, puede dirigir un programa para que lleve a cabo decisiones conforme se está ejecutando. Los que aparecen seguidamente son ejemplos de sentencias IF:

```
 :    sentencias adicionales del programa que no se indican
 :
100   IF A = B THEN 750
110   IF D > 6 THEN 280
120   IF 8 < X THEN 940
130   IF A * B <= 15.3 THEN 2040
140   IF A – D <> E THEN 3000
150   IF H < M/N THEN 305
 :    más sentencias no señaladas
```

En estas sentencias los símbolos =, >, <, >=, <=, <> significan respectivamente:

```
=     igual
>     mayor que
<     menor que
>=    mayor o igual que (=> es también aceptable)
<=    menor o igual que (=> es también aceptable)
<>    no es igual (>< es también aceptable)
```

Veamos cómo es la primera sentencia en el ejemplo anterior:

```
100   IF A = B THEN 750
```

Cuando el programa encuentra esta sentencia, examina los valores asignados en este momento a A y a B. Si el valor asignado a A es exactamente igual al valor asignado a B el programa saltará a la línea 750.

Si el valor de A no es igual al de B, el programa continua con la siguiente sentencia en secuencia. Esta sentencia es la de la línea 110.

La sentencia IF de la línea 110 comprueba cuando D es mayor que 6. Si ello sucede el programa salta a la línea 280; en caso contrario el programa va a la sentencia siguiente en secuencia. Esta sentencia está en la línea 120.

Mire las sentencias IF dadas en el ejemplo. Vd. observará que cada sentencia IF tiene un símbolo de relación escogido del conjunto =, >, <, > =, < =, < >. (Los últimos tres símbolos relacionales se forman escribiendo dos caracteres consecutivos que se hallan en el teclado de su terminal.)

Los valores señalados sobre cada lado del símbolo relacional, pueden ser nombres de celdas de la memoria teles como A, B, D, X, H; valores numéricos actuales tales como 6, 8, 15.3; o expresiones aritméticas tales como A * B, A – D, M/N.

Cuando Vd. escribe una expresión aritmética puede usar los símbolos + para significar suma; – para significar resta; * que significa multiplicación; / para significar división; y ↑ para significar exponenciación. Las expresiones aritméticas pueden usarse en LET, IF, y (como veremos posteriormente) en sentencias PRINT.

Por ejemplo, aquí tenemos una expresión aritmética en una sentencia LET:

```
200 LET P = A + B – C * D + E/F – G ↑ 3
```

El ordenador ha sido instruido para sumar A a B, restar C multiplicado por D, sumar E dividido por F, y restar G elevado al cubo. El resultado final se asigna a P.

En una sentencia IF, puede aparecer una expresión como ésta:

```
3005  IF D/F + H ↑ 5 > K * L THEN 4050
3010  LET J = W + 2
```

El ordenador es instruido para comprobar cuando D dividido por F más H elevado a la quinta potencia es mayor que K multiplicado por L. Si sucede así, se le dice al programa que salte a la sentencia de la línea 4050; si no es correcto el resultado de la comparación se le dice que vaya a la siguiente sentencia en secuencia, es decir a la 3010.

En un capítulo posterior, discutiremos con detalle cómo escribir expresiones aritméticas más complejas. Por ahora, el conocimiento adquirido le permitirá hacer los problemas y ejercicios dados en los capítulos siguientes.

GO TO en las sentencias IF

Muchos sistemas de tiempo compartido permiten situar las palabras GO TO en el lugar de THEN en las sentencias IF. Así, la sentencia IF:

```
1000 IF X = Y THEN 2000
```

puede escribirse también

```
1000 IF X = Y GO TO 2000
```

Nosotros preferimos, el uso exclusivo de THEN ya que esta es la forma standard de escribir la IF; no todos los sistemas reconocen GO TO en esa sentencia.

EJERCICIOS Y CUESTIONES

1. ¿Qué tipo de sentencia BASIC se usará cuando desea que el computador efectúe decisiones cuando ejecuta su programa?
2. En las sentencias IF, ¿cuáles son los seis símbolos relacionales que puede usar y qué significan?
3. Si la relación calculada en una sentencia IF se halla verdadera dónde irá el programa?
4. Si la relación calculada en una sentencia IF se encuentra falsa, ¿dónde irá el programa?
5. ¿Cuáles son las tres formas que toman los valores que aparecen en ambos lados de los símbolos relacionales en las sentencias IF?
6. ¿Qué cinco símbolos aritméticos puede usar en expresiones en BASIC cuando está escribiendo las sentencias LET, IF o PRINT?
7. ¿Qué palabras pueden sustituir a THEN en las sentencias IF, en algunos sistemas?
8. Verdadera o falsa. La sentencia IF escrita a continuación está correctamente escrita:

```
500 IF B > Q * R THEN GO TO 600
```

9. Verdadera o falsa. El segmento de programa escrito a continuación contiene un error de lógica

```
:
:
610 IF W = R/T THEN 620
620 LET S = T – X
:
:
```

10. Estudiar el siguiente programa y decir qué valor o valores imprimirá el ordenador.

```
10  LET X = 2
20  LET Y = 3
30  LET Z = 4
40  IF X > Y THEN 70
50  IF Y > Z THEN 100
60  PRINT Z
70  IF X > Z THEN 90
80  PRINT Z
90  PRINT X
100 PRINT Y
110 END
```

11. Si las tres primeras sentencias del programa anterior son:

```
10 LET X = 5
20 LET Y = 4
30 LET Z = 3
```

¿Qué valor o valores imprimirá el programa?

12. Si las tres primeras sentencias del programa anterior son:

```
10 LET X = 10
20 LET Y = 15
30 LET Z = 5
```

¿Qué valor o valores imprimirá el programa?

CAPITULO **5**

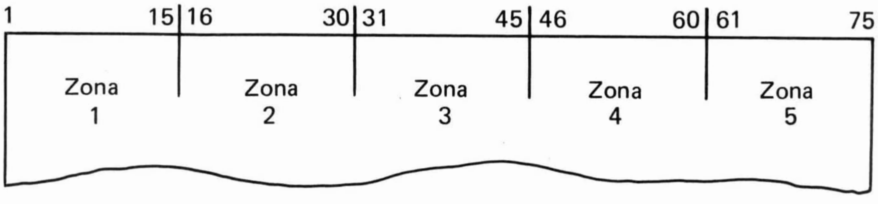
La sentencia PRINT

La sentencia PRINT se usa con diversas finalidades. Una de ellas es que el ordenador imprima los valores que previamente les han sido asignados a las celdas del computador:

Las sentencias toman esta forma:

```
        :
        :  sentencias adicionales no señaladas
        :
30      PRINT P, W, L
45      PRINT D, D3
58      PRINT M; T; V; Z
59      PRINT E1, E3; E6
```

En la línea 30 se le manda al ordenador que imprima los *últimos* valores que le fueron asignados a las celdas de la memoria P, W y L. El ordenador imprimirá dichos valores comenzando en las columnas 1, 16 y 31 del papel usado por su terminal.

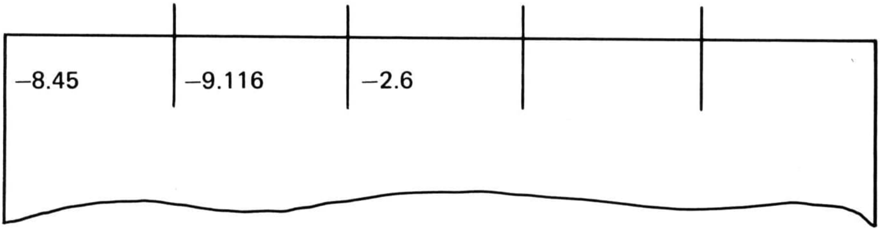


La salida del papel queda dividida en cinco zonas como aparece en la figura.

La Zona 1, se extiende desde las posiciones de impresión de la 1 hasta la 15; la Zona 2, desde la 16 hasta la 30; la Zona 3, desde la 31 hasta la 45; la Zona 4 desde la 46 hasta la 60; y la Zona 5, desde la 61 hasta la 75. (En algunos terminales, las posiciones de impresión se ex-

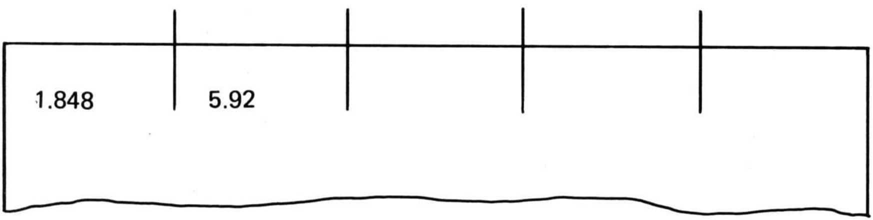
tienden solamente hasta la 72. Por consiguiente la quinta zona sería más corta que las otras.)

Las posiciones de impresión que acabamos de mencionar no aparecen de hecho impresas sobre el papel de salida. Vd. debe imaginar su existencia. En el ejemplo anterior, supongamos que los valores últimamente asignados a P, W y L fuesen –8.45, –9.116 y –2.6 respectivamente. Como respuesta PRINT P, W, L el ordenador imprimirá:



Los tres signos menos están situados en las columnas 1, 16 y 31. Si los tres valores hubiesen sido positivos, las tres posiciones de impresión mencionadas hubiesen contenido blancos.

La segunda sentencia PRINT es la 45 PRINT D, D3. Si los valores últimamente asignados a D y D3 fueran 1.848 y 5.92 respectivamente, la línea impresa sería:



Puesto que al ser los dos valores positivos, las posiciones de impresión 1 y 16 son blancos. El dígito 1 aparece en la posición de impresión 2, y el dígito 5 aparece en la posición de impresión 17.

La sentencia de impresión PRINT de la línea 58 se lee:

```
58 PRINT M; T; V; Z
```

Adviértase que el punto y coma en lugar de las comas separa los nombres de las celdas de la memoria. El punto y coma da lugar a que los valores sean impresos más próximos entre sí.

Si los valores son positivos, dos blancos separan un número de otro; si son negativos, solamente un blanco separa un número de otro.

Supongamos que los valores asignados a M, T, V y Z son −8.2, −7.6, 9.43 y −1.5 respectivamente.

El ordenador imprimirá esta línea:

```
-8.2  -7.6    9.43  -1.5
```

Las zonas anteriores se ignoran. Los valores comienzan en las posiciones de impresión 1, 6, 12 y 17 respectivamente.

La última sentencia PRINT en el ejemplo es:

```
59 PRINT E1, E3; E6
```

Si los valores de E1, E3 y E6 son −.458, .95, y −.34876 respectivamente, el ordenador imprimirá esta línea:

```
-.458          | .95  -.34876
```

El valor −.458 lo imprime en la Zona 1. El signo menos aparece en la posición de impresión 1. El valor .95 se imprime en la Zona 2. El punto decimal se imprime en la posición 17. En la sentencia PRINT, un punto y coma separa E3 y E6. El punto y coma provoca la impresión del valor de E6 siguiendo al de E3 con un solo blanco separando los dos valores. Obsérvese la coma que separa el nombre E1 del nombre E3: el valor de E1 será impreso en la Zona 1; el valor de E3 será impreso en la Zona 2.

La sentencia PRINT se usa también para indicar al ordenador que imprima uno o más mensajes literales. Veamos algunos ejemplos:

```
500 PRINT "ESTE ES UN EJEMPLO DE MENSAJE"
510 PRINT " QUE LE PEDIMOS"
520 PRINT "  QUE IMPRIMA AL ORDENADOR"
```

Obsérvese cuidadosamente que no hay ningún blanco entre las comillas y la letra T en la primera sentencia; que hay un blanco entre las co-

millas y la letra W en la segunda sentencia; y que hay dos blancos entre las comillas y la letra T en la tercera sentencia.

Cuando el ordenador imprime estas tres líneas, el carácter que sigue inmediatamente a las comillas en cada sentencia, se imprime en la posición 1 del papel de la salida. El ordenador imprimirá estas tres líneas.

```
ESTE ES UN EJEMPLO DE MENSAJE
 QUE LE PEDIMOS
  QUE IMPRIMA AL ORDENADOR
```

Obsérvese que los tres blancos que fueron indicados en las sentencias PRINT han quedado reflejados en las líneas impresas.

Se pueden dar al programa sentencias PRINT como esta:

```
650 PRINT" VALOR DE A ES"; A;" DE B ES"; B
```

Supongamos que el valor de A es 91.65 y que el de B es –546.8. El ordenador imprimirá esta línea:

```
VALOR DE A ES  91.65 DE B ES -546.8
```

Un cuidadoso examen de la sentencia PRINT indica que V en VALOR aparece en la posición de impresión 1; que hay dos blancos impresos entre la letra S en ES y el 9 de 91.65; que existe un blanco antes de la letra D en el segundo DE; y que hay un blanco entre la letra S en el segundo DE y el signo menos que precede a –546.8.

Un tercer uso de la sentencia PRINT es lograr que el ordenador imprima el resultado de un cálculo hecho dentro de la sentencia PRINT. Son ejemplos estas sentencias:

```
2500 PRINT A + B
2510 PRINT B/2,5.3 * 3, A - B
2520 PRINT A, B, B ↑ 3,946
```

Con la primera sentencia PRINT, el ordenador sumará A y B y a continuación imprimirá el resultado. Supongamos que el último valor asignado a A era 10 y que el último valor asignado a B era 2. El ordenador imprimirá esta línea:

```
 12
```

Los dígitos 1 y 2 impresos en las posiciones 2 y 3 respectivamente.

Adviértase que la sentencia PRINT puede escribirse correctamente como:

```
2500 PRINT A + B
```

pero no hubiese sido correcta si se hubiera escrito

```
2500 PRINT C = A + B
```

El resultado de un cálculo no puede asignarse a una celda de memoria en una sentencia PRINT. Pero son perfectamente admisibles las dos sentencias

```
2500 LET C = A + B
2505 PRINT C
```

La sentencia PRINT

```
2510 PRINT B/2, 5.3 * 3, A – B
```

da lugar a tres cálculos. El ordenador imprimirá:

```
 1              15.9            8
```

A y B tienen todavía los valores 10 y 2 respectivamente.

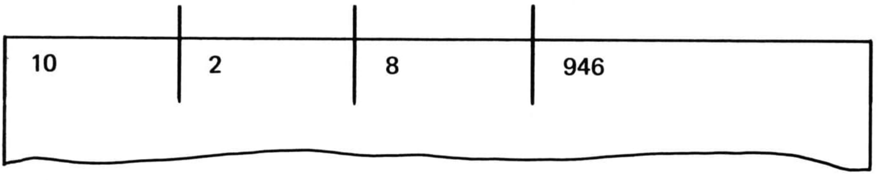
Los valores anteriores comienzan en las posiciones de impresión 2, 17 y 32 respectivamente.

La tercera sentencia de impresión

```
2520 PRINT A, B, B↑3, 946
```

da lugar a que los valores de A (10), B (2), B^3 (8), y 946 se impriman en las primeras cuatro zonas del papel de salida.

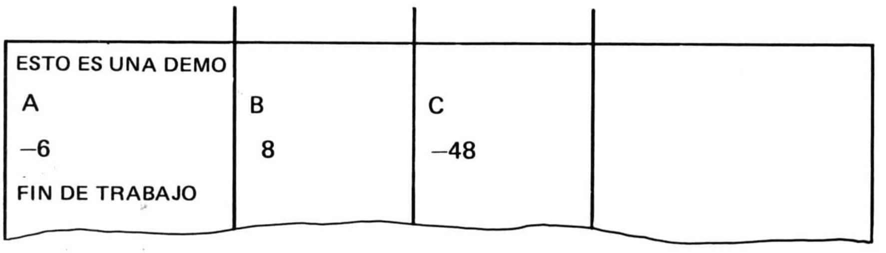
La línea aparecerá como ésta:



Los cuatro valores comienzan en las posiciones de impresión 2, 17, 32 y 47. Consideremos ahora el siguiente programa ilustrativo:

```
10 PRINT "ESTO ES UNA DEMO"
20 PRINT
30 PRINT "A", "B", "C"
40 PRINT
50 LET A = –6
60 LET B = 8
70 PRINT A, B, A * B
80 PRINT "FIN DE TRABAJO"
90 END
```

El ordenador imprimirá estas líneas:



El simple comando PRINT, como aparece en las líneas 20 y 40, da lugar a una línea en blanco.

Consideremos el ejemplo siguiente:

```
350 PRINT "ESTO ES UNA PRUEBA"
500 PRINT 346 + 925
RUN
```

El programa consta de dos únicas sentencias (350 PRINT "ESTO ES UNA PRUEBA" y 500 PRINT 346 + 925). El comando RUN le dice al sistema que *ejecute* el programa; es decir que obtenga los resultados requeridos. Obsérvese que RUN nunca está precedida por un número de línea. RUN se llama un *comando del sistema.* Un comando del sistema no es una parte del programa; sencillamente le dice al ordenador qué ha de hacer con los programas. Algunos otros comandos del sistema que estudiaremos posteriormente son SAVE, LIST, OLD, NEW, BYE y otros.

Después de que Vd. le ha dicho al ordenador que haga funcionar el programa, el ordenador lo examinará para detectar posibles errores tipográficos o incumplimiento de las reglas de programación en BASIC. Si el ordenador no encuentra ninguno, iniciará el funcionamiento del programa. Cuando sucede esto, Vd. verá estas líneas:

```
ESTO ES UNA PRUEBA
 1271
```

Cuando construye un programa Vd. escribe instrucciones llamadas *sentencias.* Existen diversas sentencias que podrían encontrarse en un programa completo en BASIC:

```
100 LET S = 0
200 FOR K = 1 to 1000
300 LET S = S + K
350 NEXT K
450 PRINT S, K
999 END
```

No se preocupe de lo que hace este programa, pero debe advertir los siguientes puntos:

1. Cada sentencia debe estar precedida por un número de línea. Los números de línea pueden escogerse arbitrariamente y pueden variar desde 1 hasta 99999.

Vd. puede elegir números de línea con intervalos grandes o pequeños entre ellos, pero deberá darlos en sucesión creciente. Si Vd. escribe sus sentencias fuera de secuencia, el ordenador las coloca por número de línea en sucesión creciente de forma automática.

2. Cada sentencia debe escribirse y a continuación dar un retorno de carro. Para escribir la primera sentencia del ejemplo, escribimos primero el número de línea (100), a continuación se pulsa la tecla para saltar espacio, y finalmente se escribe la sentencia misma (LET S = 0). Una vez terminada, se pulsa la tecla CARRIAGE RETURN.

   El sistema acepta la sentencia que se acaba de escribir. Otras sentencias del programa se escriben exactamente de la misma forma.

   Cuando ha terminado la escritura de su programa, Vd. puede desear que el ordenador comience a ejecutarlo. Para hacer esto, escriba el comando RUN.

EJERCICIOS Y CUESTIONES

1. ¿Qué tipo de sentencias en BASIC deberá usar para hacer que el ordenador imprima una línea de respuestas?
2. Verdadero o falso. La sentencia:

```
400 PRINT E, H, D
```

   provoca que los últimos valores asignados a las celdas de la memoria, E, H y D se impriman en una línea.
3. ¿Cuántas zonas de impresión existen en una línea de impresión que tiene una capacidad de 72 posiciones de impresión?
4. Dar las posiciones inicial y final de impresión de cada zona en una línea de impresión que tiene la capacidad de 72 posiciones.
5. ¿Qué señales de puntuación se usan para separar los nombres de las celdas de la memoria que aparecen en una sentencia PRINT?
6. Cómo diferirán entre sí las salidas de las tres sentencias en BASIC siguientes

```
205 PRINT L, F, P
206 PRINT L; F; P
207 PRINT L, F; P
```

7. Suponiendo que los valores de L, F y P son 2.6, −7.7 y −7.4 respectivamente. ¿En qué posición de impresión comenzará el valor de P si se usa la sentencia PRINT de la línea 205 del ejercicio anterior?
8. La misma cuestión que 7, excepto que se usa la sentencia PRINT de la línea 206.

9. La misma cuestión que 7, excepto que la sentencia PRINT usada es la de la línea 207.
10. ¿Qué señales de puntuación deberá usar en una sentencia PRINT para que se imprima el mensaje literal? ERROR IN DATA?
11. ¿A qué da lugar la sentencia

```
400 PRINT
```

12. ¿Qué hay de erróneo en la sentencia PRINT siguiente:

```
600 PRINT "VALOR DE W/ES, W
```

13. ¿Dónde se halla el error en la siguiente sentencia

```
625 PRINT W:X:Y:Z
```

14. Cómo debe Vd. escribir una única sentencia PRINT que haga que se imprima el valor de F multiplicado por L.
15. Verdadero o falso.

    La sentencia PRINT escrita a continuación es correcta:

```
820 PRINT B, T * F, 8 * 4
```

16. Decir la sentencia PRINT que hace que el ordenador imprima el mensaje:

```
TECLEAR VALORES DE R Y W
```

    Hacer que el tipo de impresión de la letra T se halle en la posición de impresión 6.

17. Estudiar el programa siguiente:

```
10 LET J = 17
20 LET K = 26
30 PRINT J + K
40 LET L = 2
50 LET K = 7
60 PRINT J + K, L
70 END
```

    ¿Qué imprimirá el ordenador con las sentencias de impresión de las líneas 30 y 60?

18. Referente a la cuestión 17. ¿En qué posición de impresión comenzará el valor de L?

19. Si desea que el ordenador imprima los valores últimamente asignados a A, B, C, D, E, F y G sobre una línea, Vd. deberá escribir:

```
1000 PRINT A,B,C,D,E,F,G
```

ó

```
1000 PRINT A;B;C;D;E;F;G
```

CAPITULO **6**

Las sentencias GO TO y END

En BASIC la sentencia IF recibe el nombre de *sentencia de transferencia condicional* porque el ordenador salta a un punto del programa dependiendo de si una condición que se consulta es verdadera o falsa. Así en

```
605 IF L<Y THEN 300
610 LET W = L + 1
```

El programa salta a la línea 300 si y sólo si el valor de L es menor que el valor de Y. En cualquier otro caso el programa continúa con la sentencia siguiente en secuencia, en este caso la línea 610.

Existe otro tipo de sentencia de transferencia en BASIC. Se trata de la *tranferencia incondicional.* Tiene la forma:

```
810 GO TO 45
```

El programa es dirigido a saltar incondicionalmente a la línea 45. Un número de línea sigue siempre a las palabras GO TO. El número de línea puede ser menor o mayor que el número de línea en la que se halla la sentencia GO TO. Por consiguiente el salto puede ser bien hacia delante o bien hacia detrás en un programa.

El tipo de sentencia final que consideraremos en este capítulo es la sentencia END. Tiene esta forma:

```
9000 END
```

En un programa solamente puede haber una sentencia END y deberá estar situada al final del programa. Por lo que ella tendrá el mayor número de línea en su programa. (En algunos sistemas en BASIC, la sentencia END es opcional. Puede omitirse.)

El ejemplo siguiente muestra cómo todos los cinco tipos de sentencias que Vd. ha aprendido aquí se pueden usar en un programa.

Supongamos que deseamos escribir un programa que imprima los resultados de

```
1 X 1
2 X 2
 :
 :
 :
 :
 :
10 X 10
```

He aquí cómo puede escribirse el programa:

```
10  PRINT "ESTE PROGRAMA REALIZA CALCULOS"
20  PRINT
30  PRINT "X", "X POR X"
40  PRINT
50  LET X = 1
60  PRINT X, X * X
70  LET X = X + 1
80  IF X > 10 THEN 100
90  GO TO 60
100 END
```

En la línea 10, el programa define la cabecera que ha de ser impresa "Este programa realiza cálculos". La sentencia PRINT "vacía" en la línea 20 proporciona sencillamente una línea de blancos entre las dos cabeceras del programa.

La sentencia PRINT en la línea 30 define las cabeceras de las columnas. Deseamos que el carácter X aparezca en la posición de impresión 1 del papel de la salida y los caracteres X POR X que aparezca su comienzo en la posición de impresión 16. La sentencia PRINT vacía de la línea 40 proporciona una línea en blanco entre las cabeceras de columna y los números de las líneas que siguen.

La sentencia LET en la línea 50 asigna el valor 1 a una celda de memoria llamada X.

La sentencia PRINT de la línea 60 hace que el programa imprima el último valor asignado a X y el resultado del cálculo X multiplicado por X. Cada vez que se hace el cálculo, el último valor asignado a X es el que se usa en la operación. Los valores de X y X multiplicados por X aparecerán comenzando en las columnas 2 y 17. Estos serán siempre va-

lores positivos; por consiguiente, las posiciones de impresión 1 y 16 nunca tendrán el signo menos impreso en ellas.

El valor de X aumenta en 1 en la línea 70. Una sentencia tal como

```
70 LET X = X + 1
```

es totalmente admisible en BASIC. Ella hace que el ordenador sume 1 al valor *viejo* de X y almacene el resultado en la misma X. La misma celda de memoria se usa para retener el valor de X. Cuando X es actualizado, el valor antiguo se destruye.

En la línea 80, el programa comprueba si el valor de X es mayor que 10. Si sucede así, el trabajo ha sido terminado y el programa se detiene. En otro caso, el programa continúa avanzando a la siguiente sentencia en secuencia. Esta sentencia está situada en la línea 90.

Cuando X es mayor que 10, el programa salta a la línea 100, la cual tiene la sentencia END. Cuando X no es mayor que 10, el programa va a la sentencia 90, la que entonces provoca un salto incondicional hacia detrás a la línea 60.

En la línea 60 se repite el ciclo ejecutado previamente. El valor de X ha crecido de forma que la línea impresa es diferente de la dada anteriormente.

Cuando el programador dirija al ordenador a ejecutar el programa escribiendo la palabra única

```
RUN
```

el programa dará esta salida:

```
ESTE PROGRAMA REALIZA CALCULOS

X               X POR X

 1              1
 2              4
 3              9
 4              16
 5              25
 6              36
 7              49
 8              64
 9              81
 10             100
```

Ahora para hacer algunas mejoras podemos lograr que el programa centre los valores de X por X cambiando la sentencia PRINT de la línea 60 a la:

```
60 PRINT X, "   "; X * X
```

Hay tres blancos entre las dos marcas de comillas. Le hemos dicho al ordenador que inserte tres blancos en la línea de impresión que comienza en la posición de impresión 16. Los blancos aparecerán en las columnas 16, 17 y 18. Los valores de X multiplicado por X comenzarán a imprimirse en la posición de impresión 20 (iniciarían su impresión en la 19 si fuesen negativos; pero no lo son). Vd. puede haber notado que el programa puede disminuir en una instrucción. Si la sentencia IF se lee:

```
80 IF X <= 10 THEN 60
```

entonces la sentencia GO TO de la línea 90 puede eliminarse. Es lo más natural borrar simplemente la sentencia en la línea 90 y dejar la sentencia 100 donde está. En lo que respecta a los números de línea, la única regla a observar es que estén siempre en sucesión creciente. Puede dejar el intervalo que desee entre los números de línea.

EJERCICIOS Y CUESTIONES

1. ¿Qué significa el término *transferencia condicional*?
2. Dar un ejemplo de una transferencia condicional.
3. ¿Qué significa el término *transferencia incondicional*?
4. Dar un ejemplo de una transferencia incondicional.
5. ¿Cuál es la última sentencia que deberá hallarse al final de cada programa en BASIC?
6. Verdadero o falso. La sentencia END debe colocarse en la línea que tiene el mayor número de línea del programa.
7. Verdadero o falso. La sentencia en BASIC que aparece debajo está escrita correctamente

```
400 GO TO 30
```

(Se supone que el programa debe tener una línea identificada con el número de línea 30.)

8. Verdadero o falso. La sentencia escrita a continuación en BASIC lo está correctamente:

```
450 GO TO 600
```

(Se supone que el programa debe de tener una línea identificada con el número de línea 600.)

9. ¿Qué hay erróneo en el siguiente programa?

```
10 LET B = 6
20 LET C = 7
30 GO TO 80
40 GO TO 70
50 PRINT "EL PROGRAMA SE PARA"
60 END
70 GO TO 50
80 PRINT B + C
90 GO TO 40
```

10. Escribir un programa que haga que el ordenador dé la salida siguiente:

```
1
2
3
4
5
FIN DE LA EJECUCION
```

Asegurarse de que su programa usa transferencias condicionales e incondicionales.

CAPITULO **7**

Las sentencias READ y DATA

Las sentencias READ y DATA van siempre asociadas en programas en BASIC. Para mostrar cómo actúan estas dos sentencias, vamos a estudiar este breve programa:

```
2000 DATA 17, 45, 61, 85, 92

2010 READ F

2020 PRINT 2 * F, F

2030 GO TO 2010

2040 END
```

La sentencia DATA proporciona una lista de valores que se usan en el programa. Estos valores son accesibles por una o más sentencias READ.

En el ejemplo, la primera sentencia que se ejecuta es READ F. El primer valor asignado a F es 17, que es el primer valor de la sentencia DATA. A continuación el programa multiplica a F por 2 e imprime a continuación el doble de F y F; es decir, el programa imprime 34 y 17. (Véase la sentencia 2020.)

A continuación el programa vuelve a la sentencia READ. Como Vd. recordará GO TO es una sentencia que obliga a que el programa salte a una sentencia distante del programa. Cuando el programa ejecuta otra vez la sentencia READ, recoge el valor siguiente todavía no usado de la sentencia DATA. Este valor se asigna a F.

El nuevo valor asignado a F es 45. Este valor nuevo sustituye al valor antiguo. Como antes, el computador imprime el doble de F y también F. A continuación el programa vuelve otra vez a la sentencia READ.

Como Vd. puede ver, el programa está en un bucle. Esto significa que el programa repite varias instrucciones una y otra vez hasta que no hay

más valores de la sentencia DATA para ser procesados. El programa imprime entonces el mensaje siguiente

```
OUT OF DATA IN 2010
```

y se detiene. Cuańdo el programa imprime el mensaje OUT OF DATA, nos dice que la sentencia READ es la causa del mensaje.

Las sentencias DATA pueden aparecer en cualquier lugar del programa con tal que ellas precedan a la sentencia END. Pueden existir varias sentencias DATA en un programa si se necesita más de una.

Las múltiples sentencias DATA se consideran lo mismo que si fueran una sentencia DATA continua y única.

Supongamos que el programa ejemplo que aparece al comienzo de este capítulo hubiese sido el siguiente:

```
2000 DATA 17, 45, 61, 85, 92
2005 DATA 84, 6, 64
2010 READ F
2015 DATA 49, 58, 67
2020 PRINT 2 * F, F
2022 DATA 3
2024 DATA 99, 95
2030 GO TO 2010
2040 END
```

El programa procedería exactamente igual a como lo hizo antes, excepto que serían procesados 14 valores de datos, en lugar de los únicos 5 anteriores. Los 14 valores serían

17, 45, 61, 85, 92, 84, 6, 64, 49, 58, 67, 3, 99, 95

Cuando el programa se detuviese imprimiría:

```
OUT OF DATA IN 2010
```

Vd. deberá colocar los valores DATA en un programa, en una forma más adecuada que la vista anteriormente. Lo mejor es colocar todas las sentencias DATA o bien al comienzo de un programa o justamente inmediatamente antes de la sentencia END.

La sentencia READ puede leer varios valores al mismo tiempo. Consideremos el programa siguiente:

```
1000 READ D, E
1010 PRINT D, E, D * E
1020 READ F, G, H
1030 PRINT F, G, F * G * H
1040 GO TO 1000
1050 DATA 3, 5, 6, 4, 7, 11, 8, 33, 14, 92, 15
1060 DATA 18, 17, 2, 6
1070 END
```

El programa asigna los valores 3 y 5 a D y E respectivamente. Luego el programa asigna 6, 4 y 7 a F, G y H respectivamente. A continuación el programa asigna 11 y 8 a D y E respectivamente; 33, 14 y 92 a F, G y H; 15 y 18 a D y E; y 17, 2 y 6 a F, G y H respectivamente. Finalmente el programa imprime

```
OUT OF DATA IN 1000
```

Con el fin de que una sentencia READ se ejecute totalmente, deberá haber al menos tantos valores sin usar todavía en la sentencia DATA como los que la sentencia READ requiera. Así, si la ejecución de la sentencia READ

```
2090 READ W, X, Y, Z
```

se va a ejecutar y solamente quedan tres valores no utilizados todavía en la sentencia DATA, el programa imprimirá también

```
OUT OF DATA IN 2090
```

Ahora pasemos a un problema práctico. Supongamos que Vd. desea sumar los valores de la sentencia DATA identificada con la línea 5000. Estudiaremos el programa que intenta efectuar el trabajo:

```
5000 DATA 46, 83, 94, 57, 66, 41, 13, 93, 88
5010 LET S = 0
5020 READ B
5030 LET S = S + B
```

```
5040 GO TO 5020
5050 PRINT S
5060 END
```

Este programa no hará el trabajo. Apenas comenzar el programa leerá diversos valores de B y se los sumará a S, obteniéndose la suma de aquéllos, pero cuando no haya más valores, el programa se detendrá e imprimirá el mensaje

```
OUT OF DATA IN 5020
```

El programa no ejecutará la sentencia de la línea 5050, ya que jamás llegará a ella.

Necesitamos un *valor ficticio* situado en la sentencia DATA; es decir, un valor que no actúe como un valor actual en el programa –un valor ficticio que únicamente indique el fin de los valores DATA.

Observemos a continuación cómo puede cambiarse el programa:

```
5000 DATA 46, 83, 94, 57, 66, 41, 13, 93, 88, 999

5010 LET S = 0

5020 READ B

5030 IF B = 999 THEN 5060

5040 LET S = S + B

5050 GO TO 5020

5060 PRINT S

5070 END
```

El valor ficticio puede ser cualquiera que un programador escoja, pero el valor elegido finalmente no deberá ser accidental o inadvertidamente igual a uno de los valores actuales que deberán ser procesados. Una sentencia IF se emplea para determinar cuándo se ha hallado el valor ficticio (o mudo).

En el momento en que se ha encontrado este valor ficticio, el programa salta a la sentencia alejada del programa. En el ejemplo, salta a 5060, donde se imprime la suma buscada.

Consideremos un problema en el que debe calcularse el promedio (valor medio o media aritmética) de varios valores. He aquí el programa:

```
2100 DATA 18, 48, 73, 63, 15, 35, 34, 999
2110 LET S = 0
2120 LET N = 0
2130 READ X
2140 IF X = 999 THEN 2180
2150 LET S = S + X
2160 LET N = N + 1
2170 GO TO 2130
2180 PRINT S/N, S, N
2190 END
```

Este programa proporciona una pequeña ventaja adicional. No solamente imprime el promedio de los valores, imprime también S, la suma de los valores y N el número de dichos valores.

NOTACION EXPONENCIAL

Frecuentemente, el ordenador de respuestas en *notación exponencial.* Las reglas que rigen cuándo el ordenador da respuestas en notación exponencial varían de un sistema a otro; sin embargo el usuario puede tener que obtener alguna experiencia con su sistema con el fin de predecir la forma que tomarán las respuestas. Si alguna de las respuestas se da en notación exponencial, tales respuestas no causan un problema especial si el usuario las entiende.

He aquí algunos ejemplos de valores exponenciales:

2.17346E 03	significa 2.17346×10^3 ó 2173.46
1.18467E 25	significa 1.18467×10^{25} ó bien 11846700000000000000000000.00

Cuando el ordenador da un valor muy grande en notación exponencial, únicamente los seis dígitos no nulos encabezando la tira de dígitos son actualmente conocidos con precisión por el sistema. Así, si Vd. le dice al ordenador que calcule 11^{10}, el computador puede almacenar el resultado en su memoria como 2.59374E 10. Esta es la salida impresa del ordenador aún cuando el valor actual de 11^{10} es 25937424601.

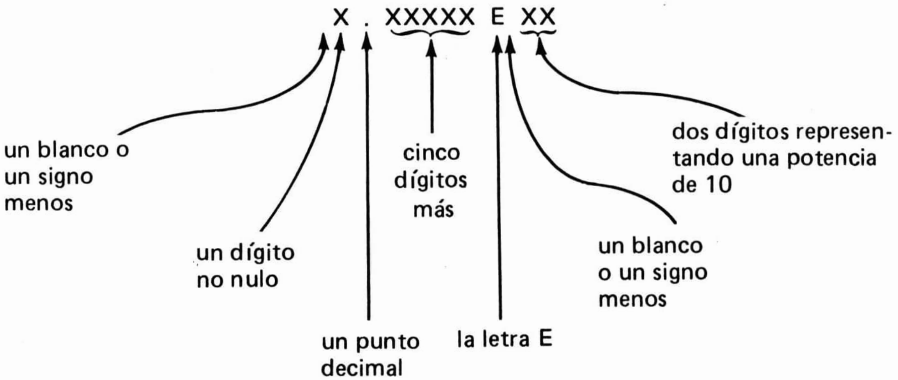
Si sus programas son de naturaleza científica, Vd. puede desear usar la notación exponencial cuando le proporciona números al ordenador

para trabajar con ellos. Vd. puede introducir aquellos números en las sentencias DATA, LET o INPUT.

Por ejemplo, puede escribir un programa de esta forma:

```
1000 DATA 2.18E5, 43, .01E4, 6.726E-4
1010 READ D
1020 LET E = (D + 2.5E6)/4
1030 PRINT D, E
1040 GO TO 1010
1050 END
```

Cuando el ordenador le da la respuesta en forma exponencial, usa un formato rígido:



Ejemplo:

−3.92765E − 03

El valor del número es -3.92765×10^{-3} ó −0.00392765.

Cuando se escribe un número con notación exponencial, se permite una gran flexibilidad. Así, puede escribir el número 181389 como

181389	1.81389E5
.181389E6	.0181389E7
.181389E + 06	1813890E − 1
.181389E 06	

y de muchas otras formas.

A continuación se muestra una tabla manual de referencia indicando las potencias de 10:

TABLA EXPONENCIAL

Exponente	*Significacion*	
−6	10^{-6}	(.000001)
−5	10^{-5}	(.00001)
−4	10^{-4}	(.0001)
−3	10^{-3}	(.001)
−2	10^{-2}	(.01)
−1	10^{-1}	(.1)
0	10^{0}	(1)
1	10^{1}	(10)
2	10^{2}	(100)
3	10^{3}	(1000)
4	10^{4}	(10000)
5	10^{5}	(100000)
6	10^{6}	(1000000)

LA SENTENCIA REM

La sentencia REM (remark, observación en castellano) es una sentencia especial que un programador puede emplear en un programa para dar mensajes a sí mismo o a algún otro observador. Se usa para decir lo que está aconteciendo en una determinada parte del programa.

```
100 DATA 3, 9, 4, 7, 8, 999
110 REM ARRIBA SE DAN LOS VALORES DATA
120 REM EL VALOR 999 ES FICTICIO
130 READ X
140 REM SE ASIGNA UN VALOR A X
150 REM Y SE COMPRUEBA SI ES FICTICIO
160 IF X =999 THEN 240
170 LET Y=X ↑ 3
180 REM SE CALCULA EL CUBO DE X
190 REM Y SE IMPRIME
200 PRINT X, Y
210 GO TO 130
```

```
220 REM EL PROGRAMA VUELVE AL READ
230 REM PARA OBTENER OTRO VALOR
240 END
```

Las sentencias REM las imprime el ordenador cuando el usuario solicita un listado de su programa, pero no tienen efecto alguno sobre la forma en que el programa resuelve o ejecuta el problema. El programa anterior hubiera dado el mismo resultado si hubiese sido escrito como éste:

```
100 DATA 3, 9, 4, 7, 8, 999
110 READ X
120 IF X =999 THEN 160
130 LET Y = X ↑ 3
140 PRINT X, Y
150 GO TO 110
160 END
```

Es una buena idea el salpicar liberalmente con sentencias REM el programa. Su existencia ayudará a cualquier persona a entender el programa mejor en un futuro.

En este texto no usaremos las sentencias REM con frecuencia porque los párrafos del texto comentan completamente los programas.

Debemos escribir los ejemplos de programas tan claros y escuetos como sea posible. Sin embargo, al programar actualmente, la única aclaración que se encuentra concerniente a algunos programas, puede estar en las sentencias REM que fueron intercaladas. Le animamos a que en sus programas use las sentencias REM con toda libertad.

SENTENCIAS PRINT TAB Y PRINT USING

Las sentencias PRINT TAB y PRINT USING son extensiones de las sentencias PRINT. Permiten gran flexibilidad para obtener la información impresa sobre el papel de la salida.

Veamos primeramente la PRINT TAB. Consideremos las diversas formas en las que puede usarse PRINT TAB:

```
10 READ A, B, C, D
20 PRINT TAB (35); A; TAB (50); B; TAB (65); C
30 READ I, J
40 PRINT TAB (I); D; TAB (J); "*"
50 DATA 5.8, 6.2, 7.9, 1.6, 25, 50
90 END
```

Este programa lee los valores 5.8, 6.2, 7.9 y 1.6 y los asigna a las celdas de memoria A, B, C y D respectivamente. A continuación imprime los valores de A, B y C comenzando en las posiciones de impresión 35, 50 y 65. (Véase la sentencia PRINT en la línea 20.) El programa lee entonces dos valores más, 25 y 50 y usa dichos valores como las posiciones de impresión en la sentencia de impresión de la línea 40. El valor de D comienza a imprimirse en la posición de impresión 25; el asterisco (*) se imprime en la posición de impresión 50. Obsérvese que los valores de A, B, C y D son positivos. Las posiciones de impresión 35, 50, 65 (y la 25 de la línea siguiente contienen blancos).

Hay 75 posiciones de impresión sobre el papel del teletipo numeradas de 1 hasta 75, ambas inclusive. (Algunos sistemas permiten llegar hasta la posición 72 únicamente.) Si manda al ordenador que imprima un valor comenzando en alguna posición de impresión dada, el sistema imprimirá el valor exactamente en dicha posición si el número es negativo; en otro caso, el sistema imprimirá el valor una posición a la derecha. Los mensajes literales se imprimen comenzando en la posición de impresión designada por Vd.

Cuando Vd. da una posición de impresión en un programa, el valor de dicha posición será un entero. Si no fuese un entero el sistema trunca la parte no entera del valor y toma la parte entera como posición de impresión.

Por ejemplo, en la sentencia

```
PRINT X, Y, TAM (50.8); Z; TAB (N); "THE END"
```

el programa imprimirá el valor de X en la primera zona del papel de salida; el valor de Y en la segunda zona; el valor de Z comenzando en la posición de impresión 50; y el mensaje THE END comenzando en la posición de impresión N. Si el valor de N es 68.6 el sistema comenzará el mensaje en la posición de impresión 68.

Los valores de TAB deben darse en sucesión creciente, en otro caso, se tendrán salidas erróneas. Por ejemplo:

```
PRINT TAB (60); A; TAB (50); B; TAB (40); C
```

dará todos los valores comenzando en la posición de impresión 60 con uno o dos blancos entre los números.

La sentencia PRINT TAB puede usarse para dibujar una curva. Supongamos que deseamos dibujar la curva $y = x^2$ desde el punto donde x es igual a 1, hasta el punto donde x es igual a 8.

Un programa que efectúe este trabajo es:

```
100 LET X = 1
110 PRINT TAB (X ↑ 2); "*"
120 LET X = X + 1
130 IF X <= 8 THEN 110
140 END
```

Los asteriscos se imprimirán en las posiciones de impresión 1, 4, 9, 16, 25, 36, 49 y 64 sobre ocho líneas. Vd. puede ampliar más el dibujo escribiendo el programa siguiente:

```
100 LET X = 1
110 PRINT TAB (X ↑ 2); "*"
120 LET X = X + .25
130 IF X <= 8 THEN 110
140 END
```

La sentencia PRINT USING puede emplearse también para situar valores sobre el papel de salida si se escribe de esta forma:

```
100 DATA 4.958, 2.71, 9.1, .053, 999

110 READ X, Y

120 IF X = 999 THEN 200

130 PRINT USING 140, X, Y

140:          #.##          #.#

150 GO TO 110

200 END
```

El programa imprimirá los valores de X e Y usando la información de la línea 140 como una guía para ubicar los dígitos. Si los puntos decimales en la línea 140 están situados en las posiciones de impresión 13 y 24, el programa imprimirá 4.96 y 2.7 sobre una línea del papel de salida con los puntos decimales situados en las posiciones de impresión 13 y 24. Imprimirá 9.10 y .05 en la siguiente línea con los puntos decimales también situados en las posiciones de impresión 13 y 24.

El signo sostenido (#) en la línea 140 señala cuántos dígitos hay que dar para cada número sobre la línea de impresión.

Los ceros aparecen a ambos lados del punto decimal, si los valores disponibles actuales son menores que los dados por los signos sostenidos.

El resultado se redondea automáticamente cuando sea necesario.

LA SENTENCIA STOP

La sentencia STOP actúa de una forma muy similar a la sentencia END, *pero ella no puede aparecer nunca al final de un programa.*

La sentencia END es la única que puede aparecer al final de un programa. Una sentencia STOP puede con frecuencia usarse en lugar de un GO TO.

Ejemplo:

```
100 DATA 9, 4
200 READ A, B
210 IF A > B THEN 240
220 PRINT "A NO ES MAYOR QUE B"
230 STOP
240 PRINT "A ES MAYOR QUE B"
250 END
```

La sentencia STOP no es imprescindible para el programador. El STOP anterior podía haber sido sustituido por GO TO 250.

EJERCICIOS Y CUESTIONES

1. ¿Qué dos tipos de sentencia en BASIC actúan asociadas en programas de BASIC?
2. ¿Cuál es la función de las sentencias DATA en programas en BASIC?
3. ¿Cuál es la función de las sentencias READ en programas en BASIC?
4. ¿Puede una sentencia READ obtener más de un valor al mismo tiempo, de una sentencia DATA?
5. ¿Dónde puede estar situada en un programa una sentencia DATA?
6. ¿Cuántas sentencias DATA puede contener un programa en BASIC?
7. ¿Pueden dos o más sentencias READ localizadas en puntos diferentes de un programa obtener valores de una sola sentencia DATA?
8. ¿Cuál es la definición del término bucle?

9. ¿Qué mensaje dará el ordenador cuando agote los valores de los datos que lee?

10. Verdadero o falso. El programa en BASIC que aparece a continuación está escrito correctamente:

```
10 READ G
20 PRINT G
30 GO TO 10
40 DATA 8, 7, 6, 5
50 END
```

11. Verdadero o falso. El programa en BASIC siguiente está correctamente escrito:

```
200 DATA 6, 4
210 DATA 7, 9
220 READ X
230 READ Y
240 DATA 4, 7
250 PRINT X, Y
260 GO TO 220
270 END
```

12. Verdadero o falso. El programa siguiente está escrito correctamente:

```
100 DATA 17, 14, 15, 11, 1, 18, 4
110 READ W
120 PRINT W
130 END
```

13. ¿Qué sucede si una sentencia READ intenta obtener más valores que los que son disponibles en una sentencia DATA?

14. ¿Cuál es la función de un valor ficticio situado en una sentencia DATA?

15. ¿Cómo se podrá escoger un valor ficticio para usarlo en una sentencia DATA?

16. He aquí una sentencia DATA para usarla en un programa:

```
100 DATA 7, 14, -3, 8, 4, 11, 1000
```

El valor 1000 es un valor ficticio. Escribir un programa que lea cada uno de los valores no ficticios, uno cada vez e imprima ese número. El programa tendrá seis líneas de impresión. Usar una sentencia IF que compruebe "el fin de los datos".

17. Usar la sentencia DATA que aparece en la cuestión 16, para escribir un valor no ficticio e imprimirlo junto con su cubo. Además su programa tendrá seis líneas de impresión. Usar la sentencia IF para comprobar el "fin de los datos".

18. Explicar lo que significa "notación exponencial".

19. ¿Qué significa la letra E en un número expresado con notación exponencial?

20. Escribir los valores 12.56, –43.67, 8.0 con notación exponencial.
21. Verdadera o falsa. La sentencia siguiente está correctamente escrita:

```
400 LET P = 92.63E6
```

22. Verdadera o falsa. La sentencia siguiente está escrita correctamente:

```
420 LET R = A + 6.36E – 3
```

23. Verdadera o falsa. La sentencia siguiente está escrita correctamente:

```
40 DATA 8, 7.6E5, 4.76, –3.6E – 2, 999
```

24. ¿Cuál es la función de la sentencia de tipo REM en BASIC?
25. ¿Qué palabras usará un programador en conexión con una sentencia PRINT que provoque un tabulado?
26. Estudiar la sentencia siguiente:

```
800 PRINT TAB (40); W
```

Suponiendo que el valor de W es –8.4 ¿en qué posición de PRINT comenzará el valor de W?
27. Estudiar la sentencia:

```
805 PRINT TAB (K); W
```

Si el valor de K es 50, ¿en qué posición de PRINT comenzará el valor de W? (el valor de W es –8.4).
28. Estudiar la sentencia:

```
810 PRINT TAB (20); F; TAB (40); "*"
```

¿En qué posición de impresión se imprimirá el asterisco (*)?
29. ¿Cuál es la función de la sentencia PRINT USING?
30. ¿Pueden existir muchos programas en BASIC que no incluyan la escritura de sentencias STOP?
31. Si un programa incluye las sentencias STOP y END, ¿puede la sentencia STOP seguir alguna vez a la sentencia END?

CAPITULO **8**

Resolución de problemas usando siete tipos de sentencias

Ahora conoce siete tipos de sentencias que funcionan en BASIC. Con solamente las sentencias READ, DATA, LET, PRINT, IF, GO TO y END puede escribir muchos programas que resuelven problemas de proceso de datos. Pasemos a considerar algunos ejemplos.

Supongamos que se necesita una tabla que señale el interés que habrá de pagar por un año al 7 1/2% de interés simple de varios préstamos variando desde $1.000 hasta $10.000 en saltos de $1.000. El programa que da esta tabla es el siguiente:

```
10  PRINT "TABLA DE INTERESES"
20  PRINT "    AL 7.5 POR CIEN"
30  PRINT

40  PRINT "CREDITO", "INTERES"
50  PRINT
60  LET L = 1000

70  PRINT L, .075 * L

80  LET L = L + 1000

90  IF L <= 10000 THEN 70
100 PRINT

110 PRINT "    FIN DE LA TABLA"
120 END
```

En este programa, la letra L significa cantidad prestada. Obsérvese que 7 1/2% deberá usarse en su forma decimal, .075. Cuando Vd. escriba RUN recibirá la salida siguiente:

```
    TABLA DE INTERESES
      AL 7.5 POR CIEN
CREDITO         INTERES

 1000            75
 2000            150
 3000            225
 4000            300
 5000            375
 6000            450
 7000            525
 8000            600
 9000            675
 10000           750

   FIN DE LA TABLA
```

La letra C en CREDITO aparece en la posición de impresión 1 del papel de salida; la letra I en INTERES aparece en la posición de impresión 16. Los valores impresos en cada línea comienzan en las posiciones de impresión 2 y 17. La letra inicial F en FIN DE LA TABLA aparece en la posición de impresión 4.

Los préstamos actuales y los valores de los intereses pueden quedar centrados más exactamente bajo sus cabeceras de columna si la línea 70 del programa se cambia por

```
70 PRINT "  "; L, "    "; .075 * L
```

Hay dos blancos entre el primer conjunto de comillas y cuatro blancos entre el segunto conjunto de comillas.

Como otro ejemplo, suponemos se le pide calcular la paga bruta de cinco empleados. Los hechos que necesita conocer sobre estos empleados aparecen en la tabla siguiente:

Número de empleado	*Horas trabajadas*	*Tasa horaria*
67046	39.5	5.50
74005	40.0	4.80
81547	38.6	5.00
83926	35.0	3.45
96149	40.0	4.25

Un programa que imprimirá la paga bruta es el siguiente:

```
100 PRINT "INFORME SALARIO BRUTO"
110 PRINT
120 PRINT "N. EMPLEADO", "HORAS TRABAJADAS",
          "TASA HORARIA", "SALARIO BRUTO"
130 PRINT
140 PRINT 67046, 39.5, 5.50, 39.5 * 5.50
150 PRINT 74005, 40.0, 4.80, 40.0 * 4.80
160 PRINT 81547, 38.6, 5.00, 38.6 * 5.00
170 PRINT 83926, 35.0, 3.45, 35.0 * 3.45
180 PRINT 96149, 40.0, 4.25, 40.0 * 4.25
```

Los valores actuales dados y los resultados de los cálculos se imprimen en cada línea. Por ejemplo después de que se imprimen las cabeceras, los valores impresos en la línea siguiente serán 67046, 39.5, 5.50 y 217.25. Estos valores comenzarán a imprimirse en las posiciones de impresión 2, 17, 32 y 47 respectivamente.

La cabecera inicial, INFORME SALARIO BRUTO, podría aparecer mejor centrada si se cambia la línea 100 por la

```
100 PRINT " ", "   "; "INFORME SALARIO BRUTO"
```

Hay un blanco entre el primer conjunto entre comillas y tres blancos en el segundo conjunto entre comillas. El programa imprime un blanco en la posición de impresión 1, a continuación se mueve a la segunda zona del papel de salida y da tres blancos delante de la cabecera del informe.

La cabecera del informe comienza por consiguiente en la posición de impresión 19. (Ocho blancos precederán a la cabecera).

Ejemplo:

Supongamos que deseamos calcular e imprimir los 20 números primeros de la serie de Fibonacci. La serie comienza así:

0, 1, 1, 2, 3, 5, 8, ...

Se observa que cada número es calculado a partir de la suma de los 2 números precedentes. Así, 3 se obtiene de 1 + 2; 5 se obtiene de 2 + 3; 8 se obtiene de 3 + 5, etc. El siguiente programa genera e imprime los números deseados:

```
10  PRINT "NUMEROS DE FIBONACCI"
20  PRINT
30  LET N = 0
40  LET A = 0
50  PRINT A
60  LET N = N + 1
70  LET B = 1
80  PRINT B
90  LET N = N + 1
100 LET C = A + B
110 PRINT C
120 LET N = N + 1
130 IF N = 20 THEN 170
140 LET A = B
150 LET B = C
160 GO TO 100
170 PRINT "SE HAN IMPRESO 20 NUMEROS DE FIBONACCI"
180 END
```

En este programa, N cuenta el número de valores de los números de Fibonacci que se han impreso. Adviértase que cuando N alcanza 20, el programa se para.

Este programa imprime un cero e incrementa el contador N en 1. A continuación imprime 1 y de nuevo aumenta al contador N en 1. A continuación, calcula 0 + 1 y asigna el resultado a C. Imprime C y nuevamente aumenta el contador en 1. El valor del contador ahora es 3.

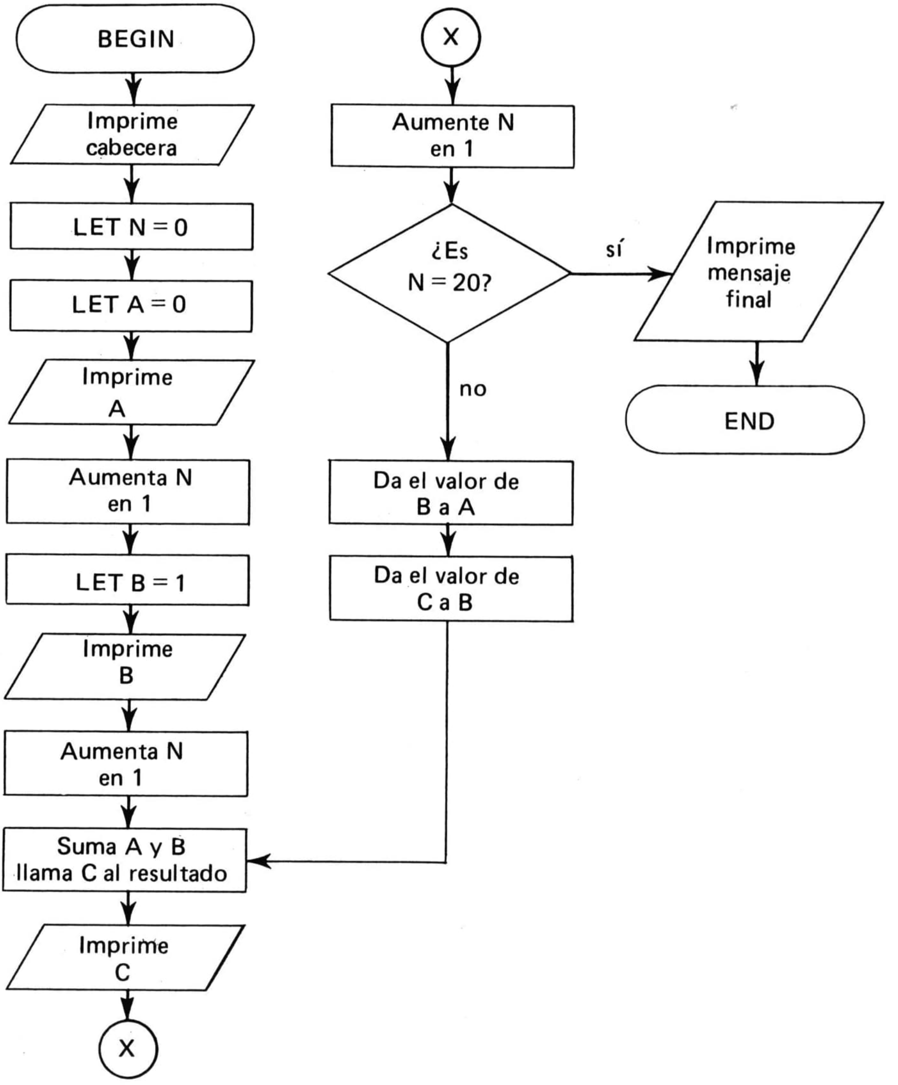
Para que el siguiente cálculo de C sea correcto, el valor de B (1) se lleva a A y el de C (1) se lleva a B. Cuando el ordenador va hacia atrás a la línea 100, el cálculo de C dará el valor correcto 2. C y N se imprimen otra vez. El procedimiento descrito en este párrafo se repite hasta que N toma el valor 20. Las sentencias

```
140 LET A = B
150 LET B = C
```

deberán entenderse bien. Estas son las sentencias que asignan el valor de B a A y el de C a B. El orden de las dos sentencias es también importante. Si las sentencias fuesen intercambiadas, entonces el valor de B sería destruido cuando el de C le fuera asignado a B. Si el valor de B había sido asignado entonces a A. A recibiría el valor de C.

Un diagrama de flujo de este programa nos ayudará a entenderlo*.

* Estudiar este diagrama de flujo para obtener una idea general de lo que es un organigrama. En el capítulo siguiente nos ocuparemos con detalle de los diagramas de flujo.



Estudiar el diagrama de flujo y seguirlo paso a paso con los números actuales que usa el programa. Convénzase que cuando el programa funciona dará esta salida:

```
NUMEROS DE FIBONACCI

 0
 1
 1
 2
 3
 5
 8
 13
 21
 34
 55
 89
 144
 233
 377
 610
 987
 1597
 2584
 4181
SE HAN IMPRESO 20 NUMEROS DE FIBONACCI
```

Un ejemplo final

Supongamos que deseamos escribir un programa que lea valores de una sentencia DATA, los sume, e imprima el promedio. Aquí está el programa que realiza la tarea:

```
1000 DATA 8, 17, 15, 4, 6, 11, 4, 8, 2, 13, 999
1010 LET S = 0
1020 LET C = 0
1030 READ V
1040 IF V = 999 THEN 1080
1050 LET S = S + V
1060 LET C = C + 1
1070 GO TO 1030
1080 PRINT S/C
1090 END
```

En el programa, los valores a ser promediados se hallan en la sentencia DATA entre la palabra DATA y el valor 999. El valor 999 no se su-

ma, actúa simplemente como un valor ficticio, señalando dónde terminan los datos.

Los nombres S y C se adoptan para *suma* y *contador* respectivamente. Observamos que cuando el valor leído, V, no sea el ficticio, 999, el programa suma el valor de V al de S y suma 1 al contador. Puesto que el número de valores en la sentencia DATA es desconocido, deberá establecerse un contador para llevar la cuenta de cuántos números son leídos y sumados a S. Una vez que se encuentra el valor ficticio, el programa salta a la sentencia que imprime S dividido por C (la suma de los valores dividida por su número).

Las sentencias de las líneas 1010 y 1020 *inicializan* S y C a cero. Estrictamente hablando, estas dos sentencias son innecesarias porque el BASIC normalmente asigna ceros a todas las celdas de la memoria antes de comenzar un programa. Sin embargo tales sentencias de inicialización constituyen una buena práctica de programar ya que el estudiante puede encontrarse en algún momento un sistema de computación que no inicialice las celdas de la memoria a cero.

No es necesario inicializar a cero variables que reciben otros valores originales. Por ejemplo, si el valor 3.1416 debe asignarse a P, se da este valor en un paso:

```
100 LET P = 3.1416
```

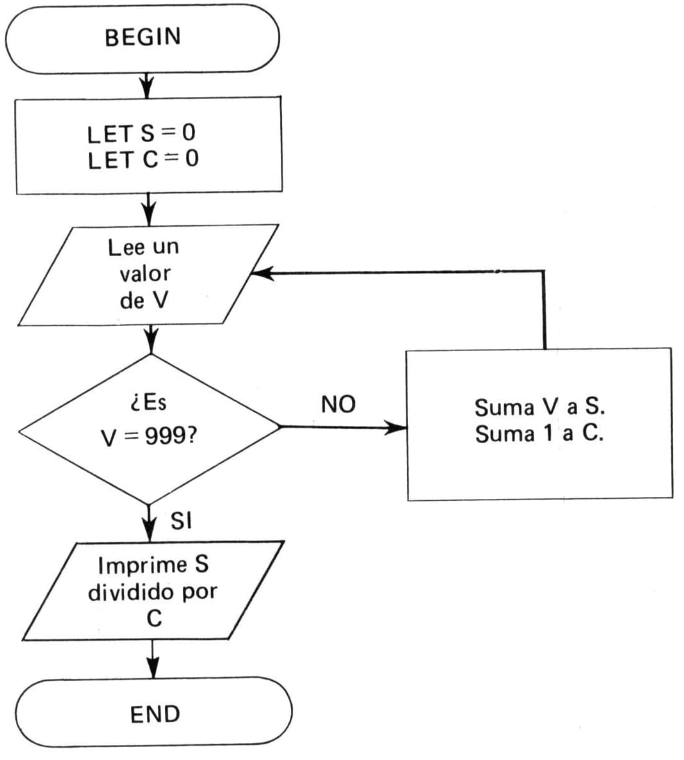
Esta asignación sustituye a cualesquiera otro valor que podía haber sido almacenado previamente en P.

En la página siguiente está el diagrama de flujo correspondiente al último ejemplo.

No daremos más detalles concernientes a los diagramas de flujo en este capítulo. Discutiremos esta importante cuestión en el siguiente capítulo.

EJERCICIOS Y CUESTIONES

1. ¿Cuáles son los siete tipos de sentencias en BASIC que han sido discutidos en el texto hasta este momento?
2. ¿Qué sentencias dan lugar a transferencias condicionales o incondicionales?
3. ¿Qué dos tipos de sentencias hacen disponibles los datos a los programas?
4. ¿Qué tipo de sentencia da lugar a que se asignen valores a celdas de memoria con nombre?
5. ¿Qué tipo de sentencia da lugar a la salida impresa?



6. ¿Dónde deberá situarse siempre una sentencia END?
7. Para este problema usar la sentencia DATA que sigue:

```
1000 DATA 9, 15, 16, 18, 21, 25, 31, 36, 999
```

(el valor 999 es ficticio e indica "fin de datos").

Escribir un programa que contraste estos valores para determinar si están en sucesión creciente. Conocemos que la observación visual puede emplearse para llevar a cabo esta determinación. Sin embargo, existen muchos casos en que los datos no son visibles al usuario y debe escribirse un programa.

En este programa, el programa ha de imprimir

```
NUMEROS EN SECUENCIA CRECIENTE
```

si está en secuencia. El programa ha de imprimir

```
NUMEROS SIN SECUENCIA
```

si el programa halla un valor fuera de secuencia.

Habiendo impreso el último mensaje, se para antes de ejecutarse otra vez.

8. Para este programa se usa la siguiente sentencia DATA:

```
300 DATA 9, 2, 6, 5, 6, 3, 4, 1, 8, 0, 0, 0
```

(los valores 0, 0, 0 al final de la sentencia DATA son ficticios y se usan para indicar "fin de datos").

Escribir un programa que lea valores que se encuentran en la sentencia DATA, tres al mismo tiempo, determine entonces cuál de los tres es el menor valor. Habiendo encontrado el menor de los tres valores, el programa imprime dicho valor, y a continuación vuelve a repetir la operación.

9. Para este programa se usa la siguiente sentencia DATA:

```
350 DATA 3, 21, 4, 9, 6, 18, 4, 1, 14, 19, 17, 1000
```

(el valor 1000 es ficticio e indica "fin de datos").

Escribir un programa que encuentre e imprima el mayor valor en la sentencia DATA.

10. Usar los mismos valores encontrados en la sentencia DATA de la cuestión 9.

Escribir un programa que encuentre el mayor y el menor valor en la sentencia, y a continuación los imprima correctamente nombrados.

11. Para este programa usar la sentencia DATA siguiente:

```
800 DATA 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 9000
```

(el valor 9000 es ficticio e indica "fin de datos").

Escribir un programa que cuente el número de ceros y el número de unos en la sentencia DATA, y a continuación imprima dichos números.

12. Para este programa usar la sentencia DATA siguiente:

```
200 DATA 8, 4, 3, 4, 0, 8, 0, 3, 4, 8, 4, 3, 2000
```

(el valor 2000 es ficticio e indica "fin de datos").

Escribir un programa que imprima cuántas veces aparece cada número en la sentencia DATA. Por ejemplo, cero aparece dos veces.

Vd. puede suponer que sabe que hay *cuatro* números diferentes en la sentencia DATA, pero lo que desconoce es qué números son.

13. Escribir un programa que calcule e imprima 10 números de Fibonacci comenzando con el 1597.

14. Escribir un programa que calcule e imprima la suma de todos los números pares desde 2 hasta 100.

CAPITULO **9**

Diagramas de flujo

Cuando un programa es sencillo, un programador puede prescindir de hacer un diagrama de flujo u organigrama antes de escribir el programa. Pero cuando un programa es moderadamente complicado o muy complicado, la preparación de un diagrama de flujo antes de escribir el programa es una exigencia casi indispensable. Muchos textos de programación ignoran los diagramas de flujo, quizás porque el arte de construirlos no puede enseñarse con facilidad. Sin embargo, se necesita algún entrenamiento formal, incluso aún en el caso de estar usando un lenguaje tan sencillo como el BASIC.

Por consiguiente lo que viene a continuación es un breve resumen de los principios concernientes a los diagramas de flujo. Insistimos al estudiante que practique los diagramas de flujo en conexión con cada programa que escriba en BASIC. Las técnicas que ha de usar se le harán familiares muy pronto, y el estudiante dejará de enfrentarse a un rompecabezas cada vez que haya de preparar diagramas de flujo.

Consideremos en primer lugar las razones por las que existen los diagramas de flujo. Son dos. La primera, porque preparando un diagrama de flujo el programador está planeando ya qué instrucciones dará finalmente al ordenador. Está desarrollando un diagrama indicando lo que él va a decirle al ordenador que haga primero, lo que haga después, etc. Conforme él desarrolla su diagrama, sus pensamientos considerando lo que el problema es actualmente y cómo va a ser resuelto comienzan a cristalizar.

El programador puede encontrarse a sí mismo borrando partes de su diagrama, cambiando y combinando otras. A veces puede advertir que su forma de atacar la resolución del problema era totalmente desacertada y puede decidir comenzarlo todo de nuevo.

Cuando un programador ha terminado un diagrama de flujo, entonces tiene un plan mostrando cómo entiende que hay que instruir al ordenador. Puede contrastar el plan suponiendo ciertos valores de entrada y a

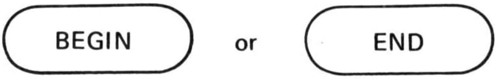
continuación seguir el diagrama para determinar lo que el computador hace con aquellos valores de entrada.

Puede advertir que el diagrama provoca que el programa tome un camino no deseado. Si esto ocurre, el diagrama se sabe que tiene un "error" en él y que ha de modificarse.

Una vez que el diagrama ha sido desarrollado y depurado, el programador puede escribir su programa con la confianza de que se ejecutará apropiadamente casi desde el primer intento. (Algunas veces hay programas con numerosos errores en ellos que requieren muchos intentos hasta que se logra que funcionen correctamente.)

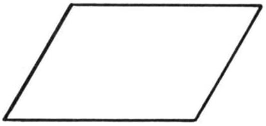
La segunda razón de por qué existen los diagramas de flujo, es el proporcionar documentación para uso ulterior. Un programador puede entender más fácilmente lo que hizo una vez o lo que hizo otro programador si existe un buen diagrama de flujo que documente el programa escrito. Justamente unos pocos momentos de cuidado extra en la documentación de un programa nos dirá con frecuencia la diferencia entre cuándo otro programador entiende fácilmente un programa o cuándo precisa un extenso período de tiempo para lograrlo.

Consideremos los símbolos utilizados en los diagramas de flujo. Estos son los principales:

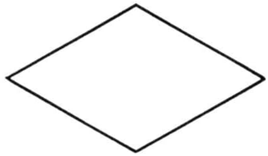


Estos símbolos indican cuándo un diagrama comienza y acaba respectivamente.

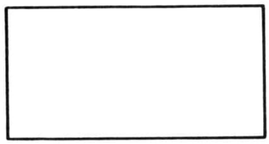
Usualmente, el símbolo BEGIN está situado en la esquina superior izquierda de la primera página de un diagrama. El símbolo END puede aparecer en cualquier parte de un diagrama.



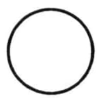
Este es el símbolo de entrada/salida y se usa para indicar la lectura de los datos de entrada o la impresión de las respuestas.



Este es el rombo de *decisión*. Se formula una pregunta que a menudo se responde *sí* ó *no*. Al menos dos líneas deben salir de un rombo de decisión. Una rotulada usualmente "*sí*", la otra "*no*".



Este es el símbolo de *asignación*. Indica qué valores son asignados a celdas de la memoria (por ejemplo, LET D = 65); ó puede indicar que un cálculo y una asignación se han hecho (por ejemplo, LET P = Q + R).

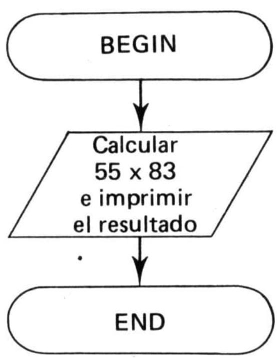


Este es el símbolo de *conexión*. Permite la conexión de una parte de un diagrama con otra.



Las flechas ligan juntamente varios símbolos del diagrama.

He aquí uno de los diagramas más sencillos que se pueden escribir:



El programa en BASIC que coincide con el diagrama anterior es

```
100 PRINT 55 * 83
110 END
```

Probemos con un diagrama de flujo más complicado. Supongamos que deseamos escribir un programa que determine cuál entre varios valores de una sentencia DATA es el mayor. Esta es la sentencia DATA:

```
100 DATA 45, 47, 92, 18, 16, 3, 14, 999
```

El último valor 999 es ficticio y simplemente indica el fin de los valores de los datos. En la página siguiente se ve el diagrama de flujo con el correspondiente programa escrito al lado.

El contenido de la sentencia DATA no se indica nunca en un diagrama de flujo porque se supone que el programa funciona prescindiendo de los datos que se emplean.

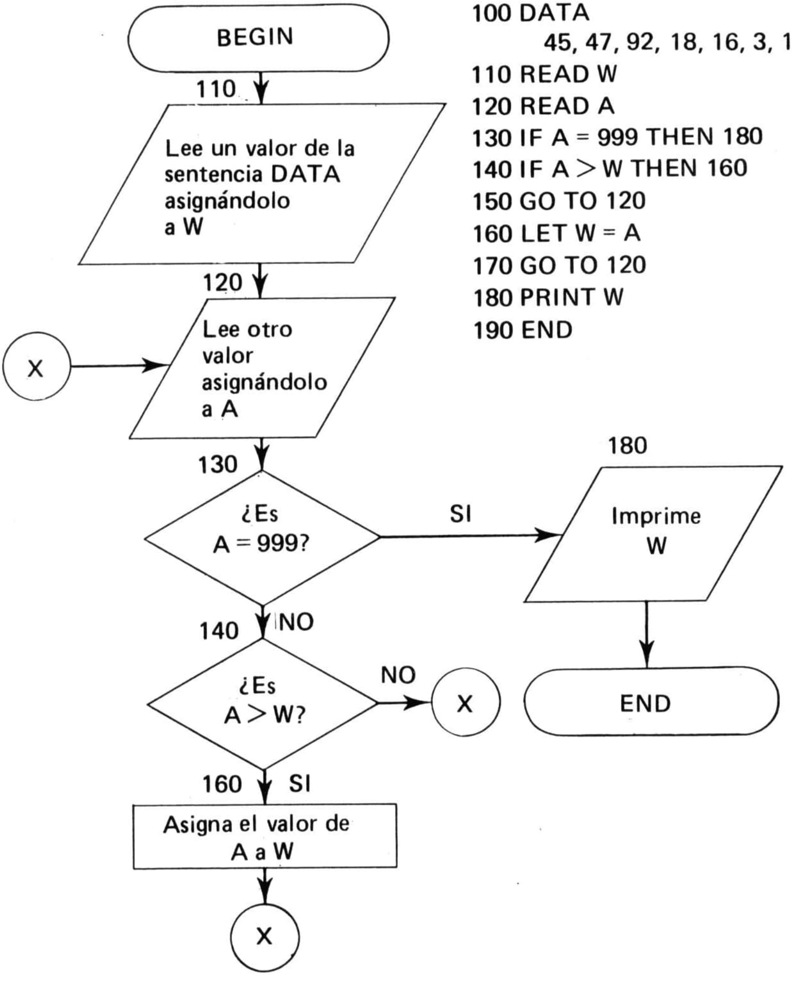
Observamos que se lee un valor de la sentencia DATA y es asignado a W. Ahora otros valores en la sentencia DATA pueden ser comparados con W. Cuando se lee un valor mayor que el de W, sustituye a W. Cuando el valor ficticio 999 se lee el programa imprime el último valor que le fue asignado a W. Este es el mayor valor encontrado en la sentencia DATA.

Si este programa fuese conservado (le indicaremos a Vd. más adelante cómo hacerlo), el programa podría usarse muchas veces en el futuro para determinar el mayor de una serie de valores. Cuando el programa funcione en el futuro, la única sentencia que deberá cambiarse será la sentencia DATA.

Observemos los números de línea situados próximos a los símbolos del diagrama. Estos números fueron situados allí después de que fue escrito el programa en BASIC. Ellos facilitan la comparación del diagrama con el programa correspondiente. Si el programa no funcionase, los números de referencia de las líneas ayudarían al programador a determinar lo que estaba incorrecto.

Habrá advertido Vd. que las flechas en un diagrama indican cómo el ordenador está siendo instruido paso a paso. Cuando se indica un rombo de decisión, la sentencia correspondiente en BASIC es la sentencia IF. El camino con el "no" en un diagrama correponde siempre a la sentencia que *sigue* a la sentencia IF.

¿Ha notado también que no es importante lo que está escrito en el interior de un símbolo del diagrama, con tal que el mensaje sea inte-



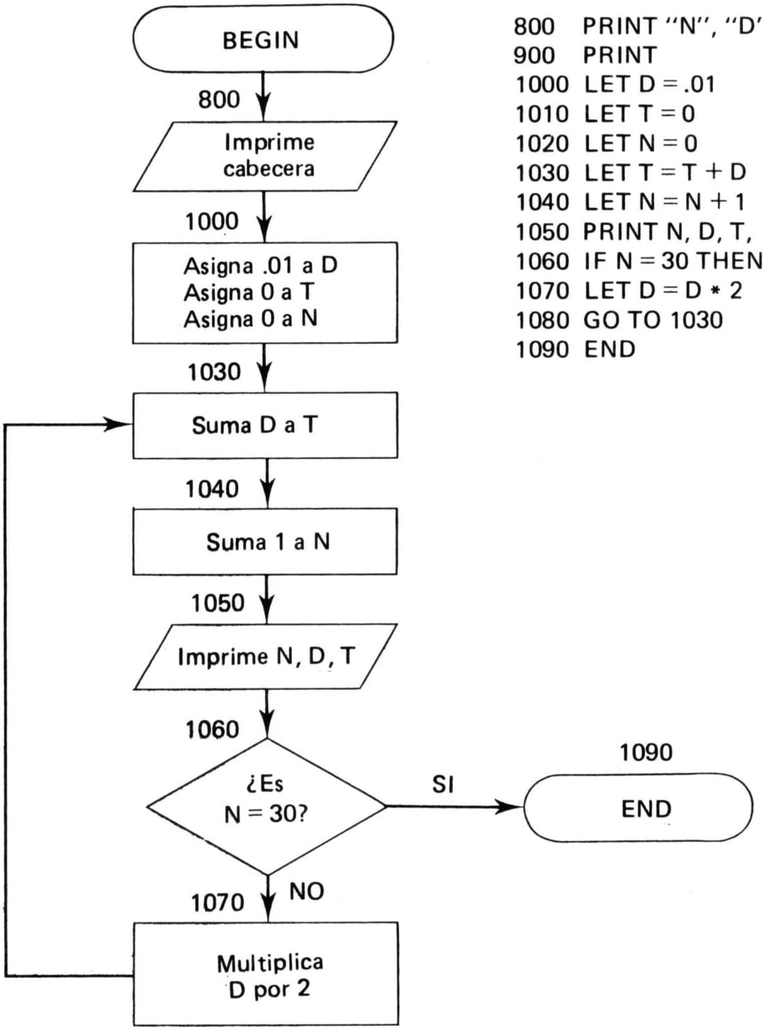

```
100 DATA
        45, 47, 92, 18, 16, 3, 14, 999
110 READ W
120 READ A
130 IF A = 999 THEN 180
140 IF A > W THEN 160
150 GO TO 120
160 LET W = A
170 GO TO 120
180 PRINT W
190 END
```

ligible y también que la sentencia correspondiente en BASIC esté correctamente escrita?

Escribimos un diagrama de flujo más, antes de abandonar este capítulo. Supongamos que tenemos que determinar la cantidad de dinero que ganará una persona si trabaja un mes (30 días) en las siguientes condiciones: trabaja el primer día por \$0.01, el segundo día por \$.02, \$.04 el tercero, \$.08 el cuarto, etc., y así hasta que trabaja los 30 días. Su paga diaria es doble de la que fue el día anterior.

El programa ha de imprimir la cantidad que él percibe cada día y la cantidad total percibida hasta la fecha para cada día del mes.

A continuación se incluye el diagrama de flujo y el programa BASIC correspondiente.




```
800  PRINT "N", "D", "T"
900  PRINT
1000 LET D = .01
1010 LET T = 0
1020 LET N = 0
1030 LET T = T + D
1040 LET N = N + 1
1050 PRINT N, D, T,
1060 IF N = 30 THEN 1090
1070 LET D = D * 2
1080 GO TO 1030
1090 END
```

En este programa D representa el valor que el trabajador ha ganado cada día (.01, .02, .04, .08, etc.), T representa el valor total que la per-

sona ha ganado hasta ese día (.01, .03, .07, .15, etc.) y N dice qué día es (1, 2, 3, 4, etc.)

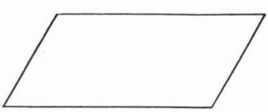
Algunas de las primeras líneas de la salida del programa son éstas:

N	D	T
1	.01	.01
2	.02	.03
3	.04	.07
4	.08	.15

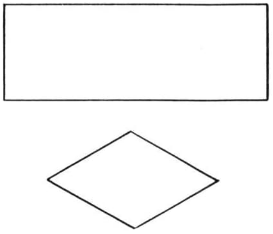
EJERCICIOS Y CUESTIONES

1. ¿Cuáles son las dos principales razones para que el programador prepare diagramas de flujo?
2. ¿Qué símbolos de diagramas de flujo indican una operación de entrada o salida?
3. ¿Qué símbolo de diagrama de flujo indica una toma de decisión?
4. ¿Qué símbolo de diagrama de flujo usa un programador para indicar dónde comienza un diagrama?
5. ¿En qué lugar de la primera página de un diagrama de flujo debería colocarse el símbolo BEGIN?
6. ¿Qué símbolo de diagrama de flujo se usa para indicar la asignación de un valor a una celda de datos nombrada?
7. ¿Qué símbolo de diagrama de flujo se llama "símbolo de conexión"?
8. Asociar los símbolos con las sentencias BASIC dadas a continuación:

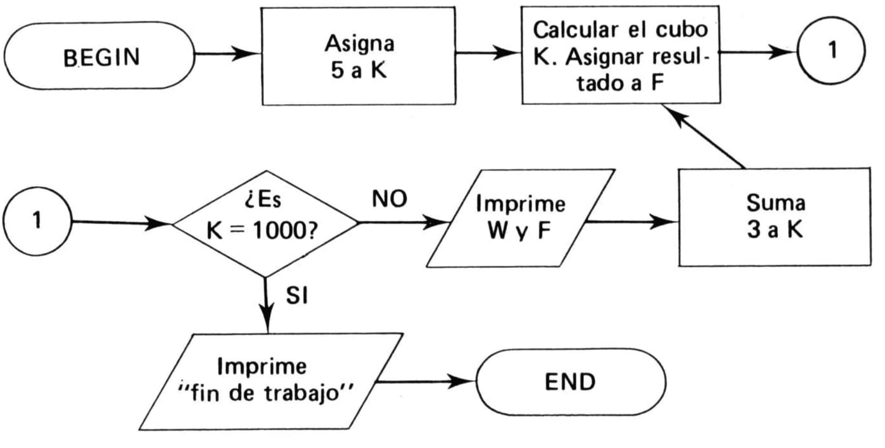
```
10  READ W
20  IF W = 6 THEN 500
30  PRINT W
40  GO TO 10
50  DATA 2, 8, 4, 6
500 END
```

Símbolo	Número/s de línea de la sentencias correspondientes
	______

Símbolo	Número/s de línea de la sentencias correspondientes



9. Escribir un programa en BASIC usando el diagrama de flujo siguiente:



10. Escribir un diagrama de flujo a partir del siguiente programa:

```
50  DATA 8, 21, 17, 6, 8, 15, 5000, 0
60  READ A, B
70  IF A = 5000 THEN 140
80  IF A > B THEN 120
90  LET F = A + B
100 PRINT A, B, F
110 GO TO 60
120 LET F = A - B
130 GO TO 100
140 PRINT " FIN DEL TRABAJO"
150 END
```

11. Escribir el diagrama de flujo que Vd. usaría para codificar un programa que halle los valores mayor y menor encontrados en una sentencia DATA. Suponer que el último valor en la sentencia DATA es 4000. Este valor es un valor ficticio usado sólo para detectar el final de los datos.

12. Escribir un diagrama de flujo a partir del siguiente programa:

```
5  PRINT "ESTE PROGRAMA CALCULA LA DEPRECIACION"
10 LET M = 20000
20 LET K = 0
30 PRINT K, M
40 LET K = K + 1
50 IF K > 10 THEN 80
60 LET M = M - M * .1
70 GO TO 30
80 PRINT "FIN DEL TRABAJO"
90 END
```

CAPITULO **10**

Aritmética

Los ordenadores son buenos realizando cálculos aritméticos. Pueden realizar cientos de miles, algunas veces millones de cálculos por segundo. Por ejemplo la sentencia en BASIC

```
105 LET F = D - E
```

puede ser realizada por algunos ordenadores en alrededor de una millonésima de segundo (un microsegundo = 10^{-6} segs.).

Cuando Vd. le indica a un computador qué cálculos ha de hacer, Vd. tiene que ser muy claro. Supongamos que Vd. desea que el ordenador le de el resultado de este cálculo:

$$\frac{41.9 + 9.5}{3.8}$$

Vd. no debe escribir la sentencia en BASIC de la siguiente forma:

```
208 LET G = 41.9 + 9.5 / 3.8
```

El ordenador lo interpretaría así

$$41.9 + \frac{9.5}{3.8}$$

que es completamente diferente de lo que Vd. desea. La forma correcta de escribir la sentencia es

```
208 LET G = (41.9 + 9.5) / 3.8
```

Los paréntesis pueden usarse para indicar cualquier cantidad que se maneja como una unidad.

Vd. debe saber que en BASIC los símbolos que sirven para hacer cálculos son

+ = suma
– = resta
/ = división
* = multiplicación
↑ = elevación a una potencia

En una serie de cálculos donde los paréntesis están ausentes, el ordenador exponencia primero. Así en la sentencia

```
2400 LET D = B * C/D + E ↑ 3 – F/G
```

el ordenador elevará primero E a la tercera potencia.

Segundas en el orden de operación son las multiplicaciones y las divisiones. El ordenador examina la sentencia de izquierda a derecha buscando estas operaciones. Así en

```
2400 LET D = B * C/D + E ↑ 3 – F/G
```

el computador multiplicará B por C, y luego divide el resultado por D. A continuación divide F por G.

Las últimas en el orden de las operaciones son las adiciones y las sustracciones. En el ejemplo señalado anteriormente el resultado de B * C/D será sumado al resultado de E ↑ 3. A continuación, el resultado de F/G será restado de la suma y el resultado final será asignado a D.

Una tabla que muestra el orden de las operaciones es ésta:

ORDEN DE LAS OPERACIONES

1. Aquéllas que están entre paréntesis
2. Exponenciación
3. Multiplicaciones y divisiones
4. Adiciones y sustracciones

Incluso dentro de los paréntesis, el computador usa la tabla anterior. Así, si hay paréntesis dentro de paréntesis, el computador actúa en los paréntesis más internos primero. A continuación sigue con las exponenciaciones, multiplicaciones, divisiones, etc.

Un número de funciones además de las antes mencionadas están incorporadas al BASIC para hacer más fácil la programación

Algunas de estas funciones son:

SIN para obtener senos trigonométricos
COS para obtener cosemos trigonométricos
SQR para obtener raíces cuadradas
ABS para obtener valores absolutos
LOG para obtener logaritmos naturales

Existen otras pocas funciones que se discutirán en el capítulo 14. He aquí algunos ejemplos de cómo estas funciones pueden usarse en sentencias en BASIC.

```
1040 LET D = SQR (L) + SQR (2.9)
1105 LET K = LOG (P/R – S)
1150 LET T = SIN (2.1) – cos (SQR (8.1))
1160 LET V = ABS (M) + (A/B – C)
```

Los números, nombres o cálculos que aparecen entre paréntesis dan los valores con los que las funciones tienen que trabajar. Así en

```
1040 LET D = SGR (L) + SQR (2.9)
```

el ordenador calcula la raíz cuadrada de cualesquiera que sea el último valor asignado a L. A continuación calcula la raíz cuadrada de 2.9.

Las dos raíces cuadradas se suman y se asignan a D.

En la sentencia

```
1105 LET K = LOG (P/R – S)
```

el ordenador da el logaritmo natural de P/R – S. Al calcular el valor de P/R – S, utilizará los valores últimamente asignados a dichos nombres.

Escribamos un programa en el que se hagan los siguientes cálculos:

$$F = \frac{\sqrt{8.1}}{A}^{3} + \sin (B)$$

Supongamos que los valores de A y B se obtienen de una sentencia DATA. He aquí el programa:

```
10 READ A, B
20 LET F = (SQR (8.1) / A) ↑ 3 + SIN (B)
30 PRINT A, B, F
40 DATA 4.5, 6.6
50 END
```

A veces puede Vd. preguntarse cuándo un conjunto de paréntesis debe incluirse en un programa. La regla simple a seguir es "en caso de duda, ponerlos". Por ejemplo, el cálculo

$$d = \frac{\frac{p}{d}}{f}$$

puede escribirse como

```
1000 LET D = P/D/F
```

ó

```
1000 LET D = (P/D)/F
```

Pero este cálculo

$$d = \frac{p}{\frac{d}{f}}$$

deberá escribirse como

```
1000 LET D = P/(D/F)
```

EJERCICIOS Y CUESTIONES

1. ¿Cuáles son los cinco símbolos aritméticos usados en BASIC? Dar su significado.
2. ¿Por qué deben a veces usarse los paréntesis al escribir sentencias de asignación?
3. En una expresión, ¿qué operación tiene el mayor nivel de prioridad cuando el ordenador hace un cálculo?
4. ¿Qué operación tiene el segundo nivel de prioridad?
5. ¿Qué operación tiene el tercer nivel de prioridad?
6. ¿Qué operación tiene el cuarto, o menor, nivel de prioridad?
7. ¿Cuáles son los nombres de las funciones incorporadas mencionadas en este capítulo?
8. Examinar esta sentencia:

   ```
   1800 LET F = (G * K) – L ↑ 3 + M/W * F
   ```

   ¿Qué cálculo se hará primero?

9. En la sentencia anterior, ¿qué calculará el ordenador a continuación?
10. ¿Cuál es la regla de "acierto seguro" referente a lo que Vd. debe hacer cuando desconoce si hay que emplear o no paréntesis en una expresión?
11. Escribir la sentencia en BASIC que coincide con el cálculo siguiente:

$$G = \frac{W - S}{P} + \sqrt{M + N}$$

12. ¿Qué errores hay en la sentencia en BASIC siguiente?

```
20 LET (G3 – H4) / F5 = L
```

13. ¿Qué valor se le asigna a Z en la sentencia siguiente?

```
LET Z = 4 – 3 ↑ (3 + 2) * 4 / (4 – 2)
```

14. Escribir dos sentencias en BASIC que coincidan con esta ecuación

$$R = \frac{-B \pm \sqrt{B^2 - 4AC}}{2A}$$

15. Dar otra solución al problema 14 usando cuatro sentencias. Las dos primeras sentencias son:

```
20 LET D = SQR (B ↑ 2 – 4 * A * C)
30 LET L = 2 * A
```

CAPITULO **11**

Comandos del sistema

Recordará probablemente que los tres comandos que ha usado no tienen número de línea. Ellos son RUN, LIST y BYE. Esta clase de comando entra dentro de la categoría general *Comandos del Sistema.*

Los comandos del sistema no son instrucciones usadas en las soluciones de los problemas. Son mensajes especiales que Vd. da al ordenador. RUN es un buen ejemplo. Esta palabra le dice al computador que Vd. ha terminado de escribir un programa y que desea ejecutarlo. Si el ordenador no encuentra errores tipográficos en su programa, él comenzará a funcionar. Que funione un programa no significa que las respuestas sean necesariamente correctas; ello significa que el ordenador no es capaz de encontrar errores de tipo lógico y por consiguiente puede hacer lo que Vd. le ha indicado que haga y nada más. Si Vd. da al ordenador instrucciones correctas pero lógicamente incorrectas, obtendrá resultados incorrectos.

Cuando Vd. entre un programa escribiendo sentencias, puede cometer errores conforme lo hace. Las correcciones pueden hacerse en cada lugar. Su programa puede tender a provocar una pequeña mezcolanza, y Vd. puede desear que el computador le de un listado en un momento dado del programa. El comando que hace que el ordenador realice esta acción es LIST. Consideremos el siguiente programa:

Ejemplo:

```
10 PRINT "PROGRAMA QUE CALCULA SI ← ENOS"
20 PRINT
30 LET X = .01
30 LET X = .1
40 PRINT X, CO ←← SIN (X)
50 IF X > 2 GO TO
45 LET X = X + .1
```

```
60 GO TO 40
70 END
50 IF X>2 GO TO 70
LIST

10 PRINT "PROGRAMA QUE CALCULA SENOS"
20 PRINT
30 LET X = 1
40 PRINT X, SIN (X)
45 LET X = X + .1
50 IF X>2 GO TO 70
60 GO TO 40
70 END
```

Un cuidadoso examen del programa original indica que:

1. La segunda línea 30 sustituye a la primera.
2. La segunda línea 50 sustituye a la primera.
3. La línea 45 se sitúa entre las líneas 40 y 50.
4. Las correcciones en las líneas 10 y 40 eliminan los errores.

Si el programador quiere conservar este programa, puede escribir

```
SAVE
```

Cuando el usuario escribe la palabra SAVE, el computador guarda el programa usando el nombre del programa que el usuario dió cuando comenzó a escribir el programa. Supongamos que en este ejemplo que el usuario desea conservar el programa con el nombre SINCMP.

Cuando un usuario ha terminado de trabajar con un programa y quiere que el ordenador comience a trabajar sobre uno nuevo, el programador escribe NEW y da el nombre del nuevo programa.

Ejemplo:

```
NEW TEST
```

El ordenador responde:

```
READY
```

Ahora el usuario puede comenzar a trabajar sobre un nuevo programa. A menos que él haya salvado el viejo programa está trabajando sobre el que no ha salvado y por lo tanto desaparece.

Cada vez que un usuario quiere trabajar con un programa viejo que él había salvado antes, puede dar el mensaje OLD y seguir con el nombre del programa. Por ejemplo, si el programador quiere recuperar un programa llamado TEST, puede escribir

```
OLD TEST
```

El computador recupera TEST y escribe

```
READY
```

El usuario puede ahora listar el programa (LIST), hacerlo funcionar (RUN), abandonarlo (UNSAVE), cambiarlo o realizar un número de otras operaciones sobre él.

Así, las palabras que hemos visto en este capítulo son ejemplos de comandos del sistema. Son estas

```
RUN
LIST
OLD
NEW
SAVE
UNSAVE
BYE
```

Como Vd. sabe, cuando una persona por primera vez contacta con un sistema de tiempo compartido, el ordenador le pregunta por su ID. También pide una respuesta al sistema y una respusta a OLD o NEW. Si la persona desea obtener un antiguo programa —uno que fue salvado en el pasado— escribe OLD y el nombre del programa bajo el cual él lo salvó. Después de que el usuario ha respondido de esta forma, el ordenador recupera el programa y escribe

```
READY
```

En lo sucesivo, la conversación entre el ordenado y el programador se realiza de la forma dicha.

El usuario de tiempo compartido deberá ser cuidadoso del hecho de que cuando él opera sobre el sistema, el ordenador pone a su disposición "una pizarra limpia" en un *espacio de trabajo temporal*. Cuales-

quiera sentencia que él escribe construye un programa en el espacio de trabajo. Cuando él da el comando SAVE, se hace una copia de este espacio de trabajo, y dicha copia se almacena en el *espacio permanente de la memoria*. Salvando algo del espacio de trabajo él no lo destruye. El usuario puede continuar añadiendo, modificando o borrando el espacio de trabajo. En cualquier momento en que el usuario quiere una copia en blanco de su espacio de trabajo, él escribe NEW otra vez y da un nuevo nombre de programa. Si desea sustituir un antiguo programa con el contenido del espacio de trabajo, él escribe REPLACE y da el antiguo nombre del programa.

Los comandos del sistema OLD y NEW destruyen el contenido de un espacio de trabajo del usuario. Así, si un programador escribe

```
NEW BETA
```

ó

```
OLD GAMMA
```

la "pizarra" en el espacio de trabajo es destruida. Cuando se use OLD, al espacio de trabajo se le da una copia del antiguo programa nombrado.

Una palabra de advertencia: Los Sistemas de tiempo compartido varían de una instalación a otra. El lector puede hallar que su sistema exige las palabras

```
RESAVE en lugar de REPLACE
RELEASE en lugar de UNSAVE
```

Puede encontrar que deba usarse el símbolo @ ó \ en lugar del símbolo ← con el fin de hacer correcciones de carácter en las sentencias. Puede también encontrar que el sistema no puede desear conocer el nombre de su programa en el momento en que él escribe NEW; deberá esperar para dar el nombre hasta que el usuario escriba SAVE.

Si la persona nunca salva un programa, el programa no es nombrado nunca.

Si el lector recuerda que los sistemas son escasamente diferentes de instalación a instalación las pequeñas diferencias entre ellos no causarán grandes inconvenientes.

He aquí una parte de una conversación que un usuario está teniendo con el ordenador. Comenzamos en el punto en que el sistema escribe OLD o NEW. Las entradas personales están subrayadas

```
OLD OR NEW–OLD INTRTN
READY
LIST
10  PRINT "ESTA ES UNA RUTINA QUE"
20  PRINT "INTEGRA EL AREA BAJO"
30  PRINT "LA CURVA Y = SIN (X)."
40  LET X = 2
50  LET W = .005
60  LET B = X + W/2
READY
UNSAVE
READY
OLD QUAD
READY
LIST
10  READ A, B, C
20  IF A = 5000 THEN 120
30  LET D = B ↑ 2 – 4 * A * C
40  IF P < 0 THEN 10
50  LET U = 4 *A
60  LET R1 = (–B + SQR (D)) / U
70  LET R2 = (–B – SQR (D)) / U
READY
50 LET U = 2 * A
80  PRINT R1, R2, A, B, C
90  GO TO 10
100  END
20  IF A = 5000 THEN 100
LIST
10  READ A, B, C
20  IF A = 5000 THEN 100
30  LET D = B ↑ 2 – 4 * A * C
40  IF P < 0 THEN 10
50  LET U = 2 * A
60  LET R1 = (–B + SQR (D)) / U
70  LET R2 = (–B – SQR (D)) / U
80  PRINT R1, R2, A, B, C
90  GO TO 10
100  END
READY
REPLACE
READY
NEW PRIMES
PRIMES ALREADY EXISTS
```

```
NEW PRIME
READY
10 PRINT "ESTE PROGRAMA CALCULA"
20 PRINT "LOS NUMEROS PRIMOS"
30 LET D=2
SAVE
READY
BYE
                                OFF AT 12:35
```

Este usuario llama al antiguo programa INTRTN; después de listarlo para ver lo que contiene, él no lo salva.

A continuación, llama QUAD, lo lista, hace algunas correcciones, lo completa, y vuelve a salver el programa bajo su antiguo nombre. Obsérvese que para resituar un programa modificado bajo su antiguo nombre, el comando del sistema REPLACE y no el de SAVE es el que se debe usar. Si el usuario hubiese incorrectamente intentado emplear SAVE, el sistema hubiese escrito

```
QUAD ALREADY EXISTS
```

Vemos un ejemplo de un mensaje de error dado por el sistema cuando el usuario prueba a crear un nuevo programa llamado PRIMES. El sistema le dice que él ya tiene un programa con este nombre. El usuario decide por tanto crear un nuevo programa llamado PRIME.

Después de que el usuario ha entrado unas pocas líneas, comprueba que es su hora de comer, entonces él salva lo que ha hecho hasta entonces y se desconecta.

EJERCICIOS Y CUESTIONES

1. ¿Qué se entiende por el término *Comando del sistema*?
2. ¿Los comandos del sistema están precedidos por números de línea?
3. ¿Qué efecto produce el comando del sistema RUN?
4. Si un programa funciona actualmente ¿puede Vd. estar seguro de que los resultados que calcula son correctos? ¿Por qué?
5. ¿Qué efectos provoca el comando del sistema LIST?
6. ¿Qué efectos provoca el comando del sistema SAVE?
7. ¿Cuál es la diferencia entre espacio de trabajo temporal y espacio permanente de la memoria?

8. ¿Qué comando del sistema elimina el espacio de trabajo temporal completamente?
9. ¿Qué hace el sistema cuando un usuario escribe OLD y da el nombre de un programa?
10. Cuando una persona salva un programa que él ha construido en el espacio de trabajo, ¿desaparece el programa del espacio de trabajo?
11. ¿Qué hace el sistema cuando una persona escribe la palabra NEW y da el nombre de un programa nuevo que él desea crear?
12. ¿Qué hace el comando del sistema UNSAVE?
13. ¿Es posible para un usuario obtener un mensaje de error cuando él escribe un comando del sistema como

    SAVE NUMANL?

    ¿Por qué?
14. ¿Es posible para un usuario obtener un mensaje de error cuando él escribe un comando del sistema como

    NEW BUDGET?

    ¿Por qué?
15. ¿En qué circunstancias deberá una persona usar el comando del sistema REPLACE en lugar de SAVE?
16. ¿Debe una persona esperar justamente un momento y considerar lo que puede acontecer después de que él escribe los comandos del sistema OLD, NEW y BYE? ¿Por qué?

CAPITULO **12**

Depuración de un programa

Un programa muy sencillo que Vd. escriba probablemente funcionará la primera vez que intente su ejecución. Programas más complejos pueden no funcionar la primera vez.

El ordenador a menudo le dice a Vd. lo que ha hecho mal. Los *mensajes de error* usualmente se limitan a errores tipográficos. Por ejemplo si Vd. entra

```
2050 PRINT D, T. L
```

cuando había realmente entendido que entraba

```
2050 PRINT D, T, L
```

El ordenador le dirá a Vd. que tiene puntuación incorrecta en la línea 2050. O si Vd. da un comando irreconocible tal como

```
4005 MOVE A TO B
```

el sistema le dirá que hay un problema de sintaxis en la línea 4005. Normalmente, Vd. no tendrá ninguna dificultad en determinar que hay errores en una sentencia o en el mismo programa cuando el ordenador da uno o más mensajes de error.

Para corregir errores, el programador ha de cambiar únicamente las sentencias incorrectas. Algunas veces tiene que insertar sentencias o borrar sentencias. Consideremos algunos ejemplos:

```
10 DATA 4, 17, 81, 41, 999, 0
20 REAT A, B
30 IF A = 999 THEN 200
40 LET C = A * B
50 PRIT A, B, C
60 GO TO 20
```

```
70 PRINT "FIN DEL TRABAJO
80 END
RUN

UNRECOGNIZABLE COMMAND IN LINE 20
UNRECOGNIZABLE COMMAND IN LINE 50
PUNTUATION ERROR IN LINE 70
MISSING LINE 200
```

Después de que el programador escribe RUN, el ordenador dio los cuatro mensajes de error señalados. El programador necesita corregir únicamente las sentencias que están mal. En consecuencia, deberá entrar estas sentencias:

```
20  READ A, B
50  PRINT A, B, C
200 PRINT "FIN DEL TRABAJO"
210 END
70
80
RUN
```

Entrando 20 READ A, B el programador hace que la nueva sentencia sustituya a la antigua. Lo mismo se aplica a la sentencia de la línea 50. Entrando 200 PRINT "FIN DEL TRABAJO", el programador proporciona la línea 200, que es exigida por la sentencia IF de la línea 30.

El proporciona también una nueva línea, la línea 210, que actúa como la nueva sentencia END. Observemos en este momento que hay dos pares de comillas en el comando PRINT.

Ahora, las líneas 70 y 80 son innecesarias y deben borrarse del programa. Para borrar líneas completas, el programador escribe los números de línea correspondientes y da vuelta al carro. El hace esta acción dos veces —una vez para la línea 70 y otra vez para la línea 80.

Cuando el programador escribe RUN, el ordenador reconoce las correcciones y el programa funciona propiamente. Estos son los resultados:

4	17	68		
81	41	3321		
FIN DEL TRABAJO				

Habiendo recibido estas respuestas, el programador puede querer un listado de este programa. El escribe

```
LIST
```

El ordenador dará la siguiente respuesta:

```
10  DATA 4, 17, 81, 41, 999, 0
20  READ A, B
30  IF A = 999 THEN 200
40  LET C = A * B
50  PRINT A, B, C
60  GO TO 20
200 PRINT "FIN DEL TRABAJO"
210 END
```

El programador puede querer ahora que el programa funcione otra vez con nuevos datos, entonces escribe

```
10 DATA 75, 44, 306, 94, 57, 83, 999, 0
RUN
```

La nueva línea 10 sustituye a la antigua. El programa da ahora esta salida:

75	44	3300		
306	94	28764		
57	83	4731		
FIN DEL TRABAJO				

¿Por qué hay un cero siguiendo el valor ficticio 999? Sencillamente porque la sentencia READ A, B debe leer siempre dos valores. Si 999 no estuviese seguido por otro valor entonces no habría suficientes números en la sentencia DATA para que se satisfaciera la sentencia READ. El ordenador daría un mensaje

```
OUT OF DATA IN 20
```

Cuando se hiciera un cuarto intento de obtener dos valores de la sentencia DATA, el programa se detendría inmediatamente y el mensaje

```
FIN DEL TRABAJO
```

no se imprimiría.

El computador es eficaz en encontrar errores tipográficos, pero no lo es en detectar errores lógicos. En el último ejemplo, si el * hubiese sido +, el ordenador no lo advertiría y no podría dar ningún mensaje de error. El estar seguro de que las respuestas impresas están dentro de ser razonables es responsabilidad del programador.

Durante la depuración de un programa, el programador puede tener que hacer alguna comprobación manual usando un calculador de pupitre. Por ejemplo, si un programa da 1,000 como respuesta al comando

```
1040 PRINT ((A * B) / (C – D)) + (E * F)
```

usando diferentes valores de A, B, C, D, E y F el programador deberá comprobar unos pocos resultados elegidos al azar usando un calculador de pupitre. Si las respuestas coinciden con las del calculador de pupitre, él puede suponer que el programa está depurado y aceptar todas las respuestas.

Las técnicas de depurado varían con arreglo a si el trabajo es para una sola vez o si es un trabajo de producción. Un trabajo de una sola vez es un programa que se necesita que funcione solamente una vez. El resuelve un problema y nunca se usa otra vez. Un trabajo de producción es un programa que deberá funcionar cada día o cada semana, etc., con datos diferentes cada vez. El trabajo de una sola vez exigirá un buen tratamiento de comprobación con un calculador de pupitre antes que el conjunto entero de resultados pueda aceptarse. Esta comprobación de forma obvia deberá hacerse únicamente una vez.

Un trabajo de producción requiere un buen tratamiento de calculador de pupitre para comprobar las primeras pocas veces que él funciona. Puesto que un programa dirigido a actuar como un trabajo de producción probablemente sustituirá a un procedimiento manual, el programa del ordenador y el procedimiento manual probablemente funcionarán en paralelo durante unas pocas iteraciones. El procedimiento manual será el único actualmente aceptado hasta que esté abundantemente claro que el procedimiento del ordenador da exactamente los mismos resultados que el procedimiento manual. El programa del computador llega entonces a ser el método primario de obtener la solución continua del problema.

Al depurar un programa, algunas veces puede ser difícil determinar dónde se hallan los errores. Supongamos, por ejemplo, que tenemos este programa:

```
100 DATA 68, 45, 147, 86, 145, 26
110 READ P, Q
120 LET F = P * Q
130 LET G = F + 35
140 LET H = G/2
150 PRINT P, Q, H
160 GO TO 110
170 END
```

El programador le ha dicho inadvertidamente al ordenador que multiplique P por Q cuando él debería haberle dicho que restase Q de P. Las respuestas son en consecuencia obviamente incorrectas. El está encontrando las respuestas en miles, cuando él esperaba que todos los resultados fuesen menores que 100.

Mirando por encima el programa, él falla en advertir el problema. (En programación, es posible examinar la misma sentencia una y otra vez y no ver sin embargo, una confusión obvia.) El programador puede recurrir a instalar sentencias adicionales PRINT en el programa de forma que él pueda seguir sus acciones paso a paso. En el ejemplo anterior, el programador podía insertar instrucciones adicionales de impresión así:

```
115 PRINT P, Q
125 PRINT F
135 PRINT G
145 PRINT H
```

Ahora un listado del programa da:

```
100 DATA 68, 45, 147, 86, 145, 26
110 READ P, Q
115 PRINT P, Q
120 LET F = P * Q
125 PRINT F
130 LET G = F + 35
135 PRINT G
140 LET H = G/2
145 PRINT H
150 PRINT P, Q, H
160 GO TO 110
170 END
```

Las sentencias que el programador escribió fueron insertadas por el ordenador en los sitios adecuados del programa. Ahora, cuando el programador escribe RUN, el programa imprimirá detalles del cálculo conforme ellos van siendo hechos. El programa verifica todavía que la sentencia READ trabaja propiamente. Aquí está la salida parcial dada por el programa:

68	45	
3060		
3095		
1547.5		
68	45	1547.5
147	86	
12642		
12677		
6338.5		
147	86	6338.5

El programador ahora indudablemente advertirá que 3060 es un valor mucho mayor que el que esperaba para F. Su atención se dirigirá a la línea 120 del programa donde indudablemente observará que LET F = = P * Q debería ser LET F = P – Q.

EJERCICIOS Y CUESTIONES

1. ¿Qué significan los términos error y depuración?
2. Verdadero o falso. Un programador puede cometer dos clases de errores cuando escribe un programa en BASIC: tipográficos y lógicos.
3. ¿Qué mensaje dará el ordenador si su sentencia en la línea 500 es la siguiente?

```
500 IF P > T THEN 700
```

4. ¿Qué mensaje dará el computador si su sentencia en la línea 150 es la siguiente?

```
50 PRINT G W T
```

5. Si Vd. tiene un error en una sentencia de su programa, ¿cómo corregirlo?
6. ¿Cómo dirigir el ordenador para que borre una sentencia de su programa?

7. Cuando un programa funciona fuera de los datos que está leyendo, ¿qué mensaje escribirá el ordenador?
8. ¿Qué se entiende por el término trabajo para una vez? ¿Cómo se diferencia de un trabajo de producción?
9. ¿Cómo debe comprobar una persona la verosimilitud de los valores dados por un trabajo de una sola vez?
10. ¿Cómo debe comprobar una persona la verosimilitud de los valores dados por un trabajo de producción?
11. ¿Qué sentencias extra PRINT en un programa, ayudan a comprobar las razones por las que un programa no da respuestas correctas?
12. ¿Qué deberá hacerse con las sentencias extra PRINT usadas para depurar cuando todas las áreas de un programa hayan sido exploradas?

CAPITULO **13**

Bucles y sentencias FOR/NEXT

En programación, Vd. querrá con frecuencia que el programa repita la ejecución de varias sentencias una y otra vez. La repetición de sentencias se llama BUCLE. El siguiente es un ejemplo de un sencillo aunque costoso bucle:

```
100 GO TO 200
200 GO TO 100
300 END
```

Este es, sin duda, el más necio de los bucles y ninguna persona en su sano juicio pretendrá que el ordenador lo realice. Un bucle tan absurdo puede terminarse solamente en una de las dos formas siguientes:

1. El usuario puede colgar el teléfono.
2. El usuario puede pulsar las teclas BREAK o INTERRUPT situadas sobre el teclado.

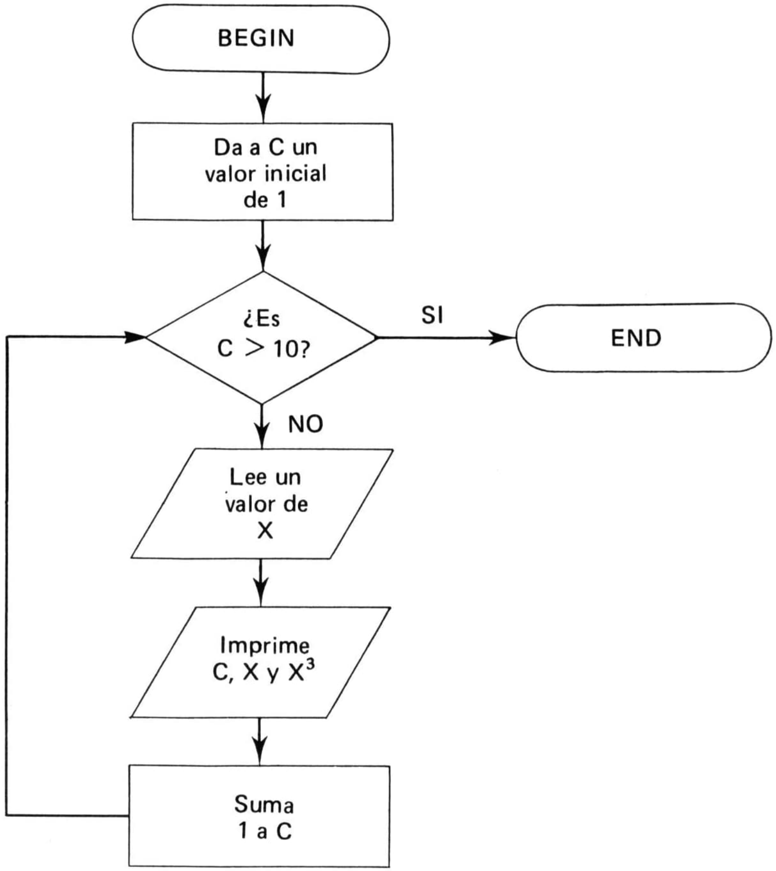
Aun cuando algunos bucles son inadvertidos y muchos de ellos innecesarios, algunos tipos de bucles están planeados deliberadamente y son muy útiles.

Supongamos por ejemplo, que Vd. tiene diez valores en una sentencia DATA que debe elevar al cubo e imprimir. Vd. podría escribir este programa:

```
1000 DATA 3, 19, 21, 33, 46, 74, 81, 89, 93, 105
1010 LET C = 1
1020 IF C > 10 THEN 1070
1030 READ X
1040 PRINT, C, X, X ↑ 3
1050 LET C = C + 1
1060 GO TO 1020
1070 END
```

La clave para entender este programa es el contador C. El tiene un valor inicial de 1. Después este valor es cambiado a 2, 3, 4, 5, 6, 7, 8, 9, 10 y 11. Cuando C es igual o menor que 10, el programa lee un valor, lo eleva al cubo, e imprime C, el valor leído y su cubo. Cuando C se incrementa a 11 la sentencia en la línea 1020 detecta este hecho y hace que el programa salte directamente a la línea 1060.

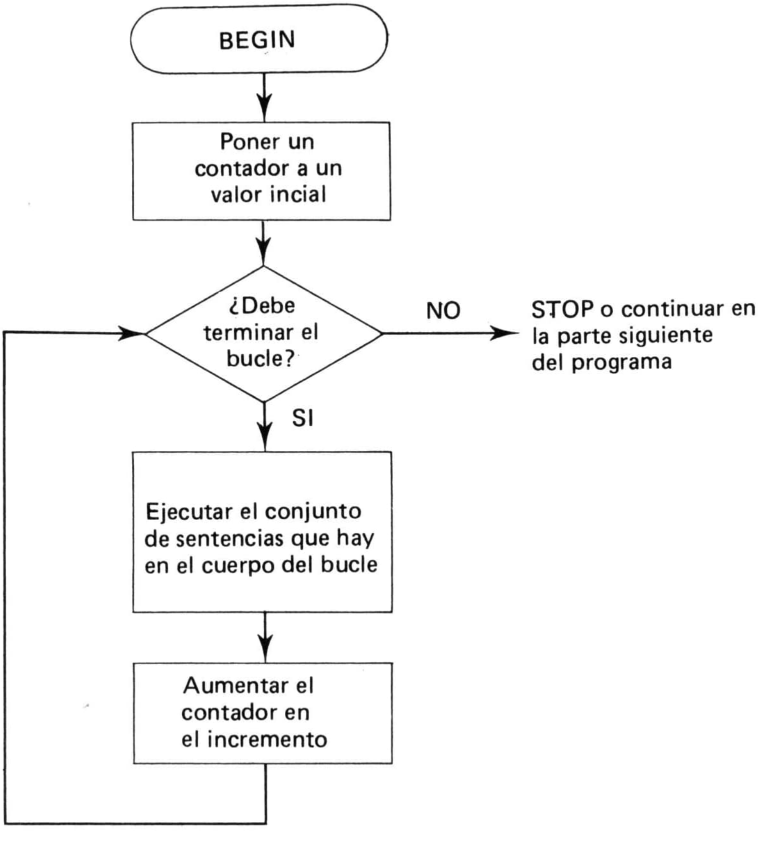
He aquí el diagrama que controla la ejecución del programa anterior:



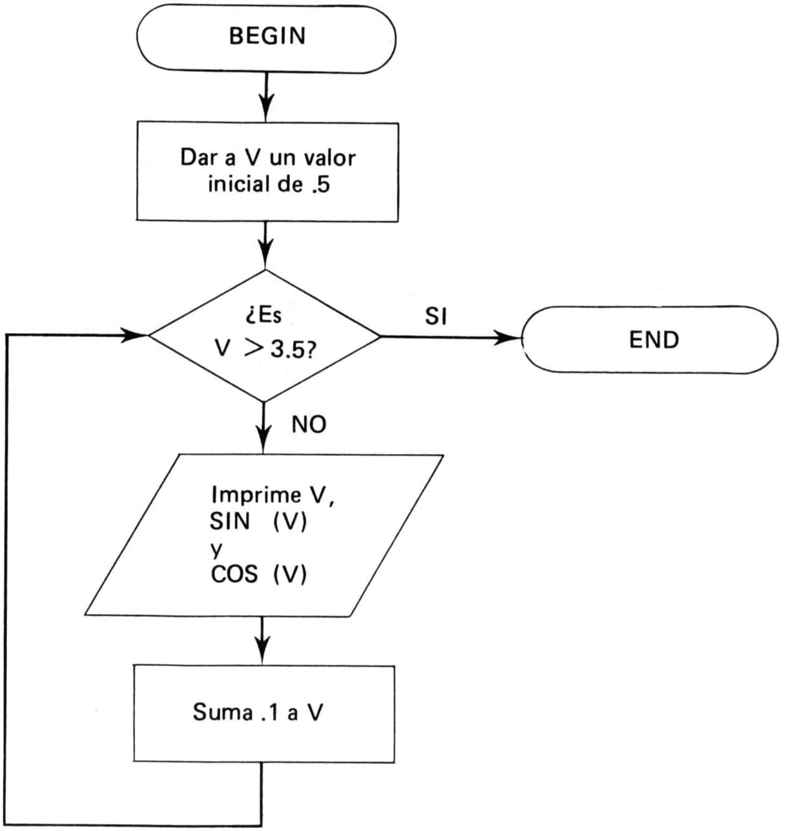
Cuando Vd. escribe RUN, el ordenador da esta salida:

1	3	27		
2	19	6859		
3	21	9261		
4	33	35937		
5	46	97336		
6	74	405224		
7	81	531441		
8	89	704969		
9	93	804357		
10	105	1157625		

Un esqueleto de diagrama que puede usarse cuando se escriben bucles es éste:



Veamos cómo funciona este diagrama de flujo otra vez para algunos problemas sencillos. Supongamos que deseamos calcular el seno y el coseno de varios valores de V variando desde .5 a 3.5 en pasos de .1. Podemos usar este diagrama:



Y el programa BASIC:

```
500 LET V = .5
510 IF V > 3.5 THEN 550
520 PRINT V, SIN (V), COS (V)
530 LET V = V + .1
540 GO TO 510
550 END
```

Supongamos que ahora queremos calcular el 6% de todos los valores comprendidos entre 10,000 y 1,000 en pasos de 500. Empleamos el diagrama anterior (apenas modificado), obteniendo este programa:

```
600  LET P = 10000
700  IF P < 1000 THEN 2000
800  PRINT P, .06 * P
900  LET P = P – 500
1000 GO TO 700
2000 END
```

Aun cuando hemos dado únicamente una sola sentencia que ha de ser ejecutada en los cuerpos de los bucles de los ejemplos anteriores, podían haberse dado varias sentencias. El programador puede situar tantas sentencias en el bucle como las que necesite.

Dos sentencias BASIC especiales pueden usarse cuando hay que codificar los bucles. Ellas son FOR y NEXT. La sentencia FOR da un valor inicial a un contador y comprueba si ha terminado; la sentencia NEXT aumenta el contador. He aquí un ejemplo:

```
400 FOR N = 1 TO 100
410 PRINT "ESTO ES UNA PRUEBA"
420 NEXT N
430 END
```

En este programa, a N se le da un valor inicial de 1. N es entonces comprobado para determinar cuándo es mayor que 100. Si sucede así el programa salta a la sentencia inmediatamente siguiente a la línea que contiene la sentencia NEXT N. Si N no es mayor que 100 el programa va a la sentencia siguiente a la sentencia FOR.

En la sentencia NEXT, el valor de N se aumenta en 1 y el programa se dirige automáticamente tras la sentencia FOR donde el bucle es otra vez comprobado para su posible terminación.

En este ejemplo, que sencillamente imprime ESTO ES UNA PRUEBA 100 veces, el nombre del contador es N; pero podría haber sido cualquier otro nombre admisible en BASIC.

Estudiemos este ejemplo:

```
300 FOR D = 10000 TO 1000 STEP –500
350 PRINT D, .06 * D
360 NEXT D
370 END
```

La sentencia FOR indica un tamaño de paso de –500. Estos significa que D cambia por incrementos de –500. El tamaño del paso es negativo, como debe ser si D ha de ir de un número mayor a un número menor. Cuando falta el tamaño del paso, el ordenador emplea siempre un tamaño de paso de 1.

Cuando el tamaño del paso es positivo, la condición de terminación se produce cuando el valor del contador es mayor que el valor de terminación dado. Cuando el tamaño del paso es negativo, la condición de terminación se presenta cuando el valor del contador es menor que el valor de terminación.

Las sentencias FOR pueden escribirse de varias formas. He aquí algunos ejemplos:

```
FOR Q = 10 TO 20000
FOR P6 = 3 TO 10 STEP .001
FOR F = –10 TO –4 STEP .1
FOR G = –20 TO 10 STEP 2
FOR B3 = 50 TO –50 STEP –2
FOR X = A TO B STEP C
FOR Y = P – 5 TO A * L STEP (M + N)/2
```

Como Vd. puede ver, el nombre del contador en una sentencia FOR puede ser cualquier nombre deseado legal en BASIC. Los valores del comienzo, final y tamaño del paso pueden ser literales numéricos, nombres o expresiones. Los nombres deberán por consiguiente haber sido valores dados previamente en el programa.

Un contador en una sentencia FOR puede actuar como un simple contador sin actuar como un valor actual dentro del cuerpo del bucle; o el contador puede actuar como un contador y como un valor actual. Observemos que cuando nosotros le dijimos al ordenador que imprimiese ESTO ES UNA PRUEBA 100 veces, el contador N no hizo nada excepto contar; cuando nosotros le dijimos al ordenador que calculase las cantidades de interés para valores entre 10,000 y 1,000, el contador D actuó como un contador y como un valor actual en el cuerpo del bucle.

Es admisible el poner bucles dentro de bucles. Consideremos este ejemplo:

```
1000 FOR X = 5 TO 100
1010 FOR Y = 1 TO 100
1020 PRINT X, Y, X * Y
1030 NEXT Y
```

```
1040 NEXT X
1050 END
```

Este programa imprime

```
5  1   5
5  2  10
5  3  15
```

etcétera.

Los números en cada línea representan los valores de X, Y y el producto de X por Y. Observemos que X toma el valor 5 mientras los valores de Y varían de 1 hasta 100. Entonces el valor de X se hace 6 y conserva dicho valor mientras los valores de Y varían otra vez de 1 a 100.

Cuando termina el programa, habrá dado todos los productos posibles de X por Y con X variando de 5 a 100 en pasos de 1 e Y variando de 1 a 100 en pasos de 1. Tendríamos 9.600 productos en total.

Es importante recalcar que un contador en un bucle nunca supera al valor máximo dado cuando los valores del contador son crecientes, ni un contador será menor que el valor mínimo cuando los valores del contador son decrecientes. Así en un bucle controlado por

```
100 FOR L = 2 TO 11 STEP 3
```

El último valor del contador usado en el bucle es 11. Los valores de L serán 2, 5, 8, 11. En

```
200 FOR M = 3 TO 12 STEP 4
```

Los valores de M usados actualmente serán 3, 7, 11. En

```
300 FOR P = 10 TO −3 STEP −2
```

los valores de P usados actualmente serán 10, 8, 6, 4, 2, 0, −2.

EJERCICIOS Y CUESTIONES

1. ¿Qué significa el término bucle?
2. ¿Son "malos" todos los bucles? Explicarlo.
3. ¿Cuál es la función del contador en la construcción de un bucle?
4. Verdadero o falso. El contador empleado en un bucle puede ser cualquier nombre lícito en BASIC.

5. En el diagrama de flujo de la estructura standard de un bucle, ¿dónde es consultado el contador?
6. ¿Qué significa el término tamaño del paso (salto)?
7. Si el tamaño del salto es positivo ¿cómo es consultada la condición de terminación de un bucle?
8. Si el tamaño del salto es negativo ¿cómo se consulta la condición de terminación de un bucle?
9. Estudiar el programa:

```
200 LET F = 1
210 IF F > 200 THEN 260
220 READ G, H
230 PRINT G * H, F
240 LET F = F + 1
250 GO TO 210
260 END
```

¿Qué línea ilustra la iniciación del contador de un bucle?

10. Usando el mismo programa que en la cuestión 9, hallar la línea que ilustra la consulta de un contador.
11. En el programa dado en la cuestión 9 ¿qué dos sentencias constituyen el cuerpo del bucle?
12. En el programa de la cuestión 9 ¿qué sentencia da una ilustración de "aumentar un contador"?
13. ¿Cuántas sentencias puede usar un programa en el cuerpo de un bucle?
14. ¿Qué dos sentencias en BASIC automáticamente escriben un bucle?
15. ¿Cuál es la función de la sentencia FOR en una sentencia BASIC?
16. ¿Cuál es la función de la sentencia NEXT en un programa en BASIC?
17. Verdadera o falsa. La sentencia FOR que viene a continuación está escrita correctamente

```
4000 FOR M = 1 TO 300, STEP A * C
```

18. Situar la letra C a continuación de todas las sentencias FOR que son correctas; situar la letra I para las que están mal (son sentencias independientes –no forman parte de ningún programa):

```
_______ 400 FOR P = 1 TO 40 STEP 3

_______ 450 FOR Q = 1 TO K STEP F

_______ 500 FOR E = –40 TO –80
```

______ 550 FOR G = H TO I STEP J

______ 600 DO 680 K = 1, 77

______ 650 FOR N = 1 TO 40 STEP –2

19. ¿Es posible situar un conjunto FOR/NEXT dentro de otro conjunto FOR/NEXT?
20. ¿Cuántos productos dará el ordenador en respuesta a este programa?

```
10 FOR G = 10 TO 20
15 FOR H = –6 TO –10 STEP –1
20 PRINT, G, H, G * H
30 NEXT H
40 NEXT G
50 END
```

21. Si el tamaño de salto de una sentencia FOR es negativo, ¿cómo debe el ordenador consulta al contador para determinar que la tarea está terminada?
22. En el siguiente programa ¿qué valores tomará el contador y usará actualmente en el cuerpo del bucle?

```
1500 FOR W = 1 TO 100 STEP 2
1505 PRINT "JOHN P. JENKINS"
1510 NEXT W
1520 END
```

23. Escribir un programa que calcule X^2, X^3 y X^4 para todos los valores de X desde 10 hasta 25 en saltos de 1. Hacer que el ordenador imprima la respuesta.
24. Escribir un programa que calcule

$$2 + 4 + 6 \ldots + 24$$

Hacer que el programa imprima las respuestas.

CAPITULO **14**

Las funciones INT y RND

En los pasados capítulos Vd. ha visto que existen varias funciones que se construyen y que están disponibles para su uso. Estas son SIN, COS, LOG y SQR. Existen unas pocas más que un programador puede desear utilizar. Dos de ellas son INT Y RND.

La función INT (en castellano llamada parte entera) da el mayor valor entero de una expresión dada. Por ejemplo en

```
400 LET W = INT (25.6)
```

el valor asignado a W es 25. La función INT no redondea. Toda fracción en el valor se borra. He aquí algunos ejemplos adicionales:

```
420 LET A = 45.8

430 LET B = 16.3

440 LET C = 8.1

450 LET D = –8.4

460 LET E = INT (B)

470 LET F = INT (C)

480 LET G = INT (A + C)

490 LET H = INT (B + D)

490 LET J = INT (A + B)

500 LET K = INT (D)
```

En este segmento del programa el valor asignado a E es 16; el asignado a F es 8; a G, 53; a H, 7; a J, 62; y a K, −9. Este último resultado puede sorprenderle. Los matemáticos definen el mayor entero de los números negativos, como siendo el valor entero siguiente en la dirección negativa. Así el valor entero de −8.8 es −9, y el valor entero de −8.1 es también −9.

La función INT puede usarse para redondear valores dentro de ciertos límites. Supongamos que deseamos redondear V al entero más próximo. Sumamos .5 a V y a continuación tomamos la parte entera del resultado.

Ejemplo:

```
100 LET A = INT (V + .5)
```

Si el valor de V es 4.3, el valor asignado a A será 4; si el valor de V es 4.7 el valor asignado a A será 5.

V puede redondearse con un número deseado de dígitos. Usamos esta relación:

```
200 LET A = INT (V * T + .5)/T
```

donde T es una potencia de 10; es decir T puede ser $10^0 = 1$; $10^1 = 10$; $10^2 = 100$, etc.

Supongamos que V tiene el valor 5.634 y deseamos redondearlo a dos cifras decimales. El valor de T sería 100 (10^2). Ahora

$$5.634 \times 100 = 563.4$$

Sumando .5 obtenemos 563.9. La parte entera es 563; y 563 dividido por 100 da 5.63. Si V hubiese sido 5.636, el resultado hubiese sido 5.64.

INT puede usarse para determinar cuándo un entero es par o impar. Usamos la sentencia IF:

```
300 IF INT (N/2) * 2 = N THEN 400
350 siguiente sentencia en el programa
```

Si el valor que está siendo consultado, N, es par, el programa saltará a la sentencia 400; en otro caso irá a la siguiente sentencia en secuencia.

La función RND da un valor aleatorio comprendido entre 0 y 1. (El valor 0 puede ser dado actualmente, pero el número aleatorio nunca será exactamente igual a 1.)

Consideremos este ejemplo:

```
110 LET D = RND (X)
```

El ordenador daría algún número a D, un número que no era conocido por el programador cuando él escribió el programa. Este número podría ser

```
.369485
.842314
.006394
.000000
.999999
.500000
```

o alguno de muchos otros números. Si la línea 110 está dentro de un bucle, el ordenador daría muchos valores a D. Por ejemplo, el siguiente es un programa que da 2000 números aleatorios.

```
1000 FOR F = 1 TO 2000
1100 PRINT RND (X)
1200 NEXT F
1300 END
```

Cada número dentro del intervalo definido anteriormente tiene una positibilidad al azar como cualquier otro número de ser impreso en cada línea. Puesto que cada número tiene igual posibilidad al azar que cualquier otro número, el número que se imprime en el momento actual se dice que es un *número aleatorio*. Así, él aparecerá como si los números que actualmente son dados procedieran de la extracción de una caja que contuviera todos los números posibles. El mismo número podía ser seleccionado dos o más veces, pero tal posibilidad es remota.

Observemos que la función RND exige que entre paréntesis se sitúe algún valor. Este valor puede sencillamente ser X como se indica. Si Vd. tiene actualmente un valor X en su programa, no habrá conflicto. No es cuestión cuando X no tiene o tiene algún otro uso en el programa. Las dos acciones son independientes una de otra. Por consiguiente esta sentencia está correctamente escrita:

```
1500 LET E = RND (X)
```

cuando X es o no usado en el programa para otros fines.

En el ejemplo anterior, Vd. podía haber repetido los 2000 valores aleatorios, sencillamente escribiendo RUN otra vez. Esta modalidad es

útil para propósitos de depuración, pero el usuario puede desear números totalmente imprevisibles.

Para obtener números imprevisibles que no pueden repetirse, el usuario debe escribir RANDOM en la primera línea de su programa.

Ejemplo:

```
1000 RANDOM
1010 FOR N = 1 TO 2000
1020 PRINT RND (X)
1030 NEXT N
1040 END
```

El ordenador dará un conjunto de 2,000 valores aleatorios. El programa se ejecutará una y otra vez, los 2,000 valores del conjunto serían diferentes.

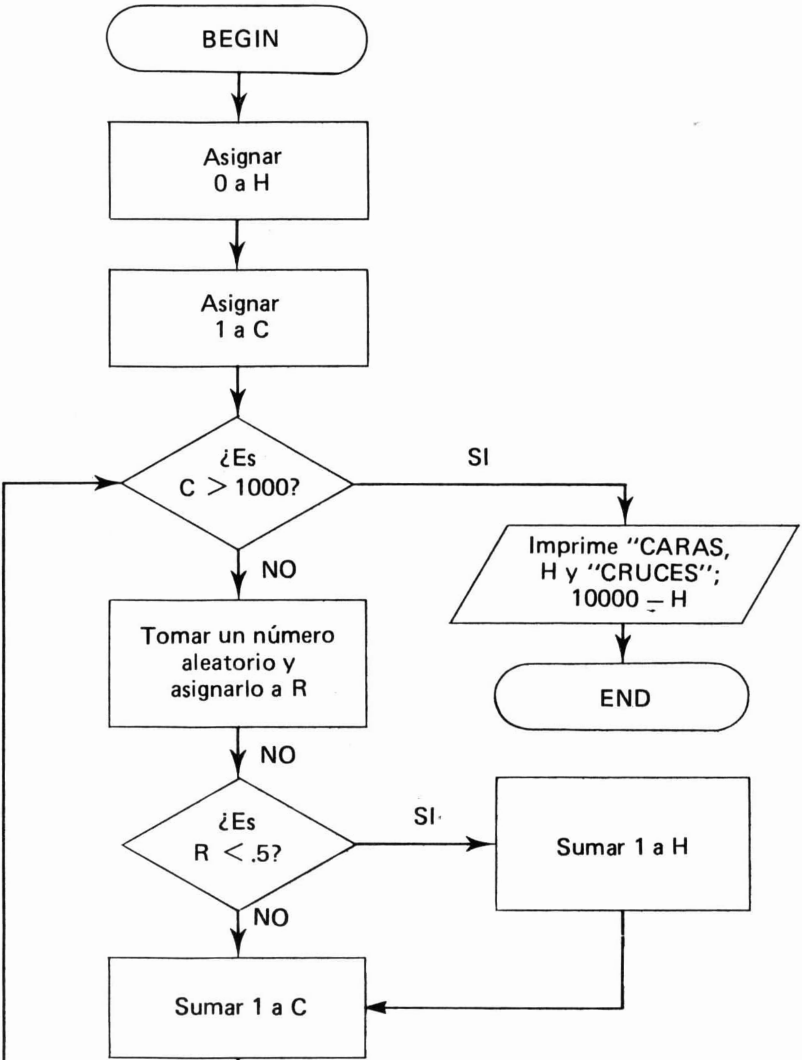
Los números aleatorios se usan para simular sucesos que tienen su lugar en la vida real. Supongamos que deseamos saber qué acontece cuando una moneda es lanzada 10,000 veces. Deseamos conocer aproximadamente cuántas caras obtendremos y cúantas cruces.

Simulamos el lanzamiento indicando el ordenador que nos dé 10,000 números aleatorios. Añadiremos que si cada número entre 0 y .5 (incluyendo 0 pero no .5) el número indica que ha aparecido una cara y si el número está comprendido entre .5 y 1 (incluyendo .5 pero no 1), el número indicará que una cruz ha aparecido.

El programa es

```
90  RANDOM
100 LET H = 0
110 LET C = 1
120 IF C > 10000 THEN 300
130 LET R = RND (X)
140 IF R < .5 THEN 200
150 LET C = C + 1
160 GO TO 120
200 LET H = H + 1
210 GO TO 150
300 PRINT "CARAS"; H; "CRUCES"; 10000 – H
310 END
```

He aquí el diagrama que podemos escribir del programa anterior:



Si estudia este programa, verá que H lleva la cuenta de cuántas caras fueron obtenidas. No hay un contador separado para las cruces puesto que después que se han hecho 10,000 lanzamientos, el número de cruces que se obtuvieron puede ser calculado restando el número de caras de 10,000. En el programa, H cuenta el número de caras ("Heads") ob-

tenido y C cuenta el número de cruces. Cuando C alcanza 10.001 (10.000 tiradas simuladas), la consulta ha quedado terminada. (Un funcionamiento actual de este programa dio 5.056 caras y 4.944 cruces.)

En consecuencia, una sola ejecución de este programa puede dar lugar a resultados ambiguos. El programa deberá funcionar varias veces. Una forma sencilla de efectuar este objetivo es insertar estas sentencias:

```
95  FOR G = 1 TO 10
305 NEXT G
```

El programa realizará ahora 10 simulaciones y dará el resultado para cada una. Un examen de los resultados nos dará una clara indicación de lo que acontecería en la vida real si alguien fuese lo suficientemente ambicioso como para lanzar una moneda 10.000 veces.

A veces, el número aleatorio dado por la función RND no es muy útil en su forma cruda u original. Puede ser deseable cambiarle a un entero dentro de algún intervalo dado.

Para efectuarlo se pueden combinar INT y RND de la forma siguiente:

```
100 LET P = INT (RND (X)) * 15) + 6
```

El número aleatorio dado por RND (X) se convierte en algún valor entero entre 6 y 20 ambos inclusives. El 15 indica ahora cuántos números están dentro del intervalo y el 6 indica el número que comienza el intervalo. Si RND (X) da .3, el número asignado a P será 10; si RND (X) da .001 el número asignado a P será 6; finalmente si RND (X) da .999, el número asignado a P será 20.

Para consulta, he aquí la regla:

```
1000 LET N = INT (RND (X) * R) + B
```

N es el número entero aleatorio deseado.

R representa el número de valores diferentes posibles dentro del intervalo.

B representa el valor de comienzo en el intervalo.

He aquí otro ejemplo que indica cómo los números aleatorios pueden ser útiles. Supongamos una estación de TV operando programa del tipo "llamando con premio" señalando que desea obtener los 10 números aleatorios de teléfono de 7 dígitos que comienzan con 371. El programa para obtenerlos es:

```
10 FOR K = 1 TO 10
20 PRINT INT (RND (X) * 9999) + 3710001
30 NEXT K
40 END
```

El programa de los números telefónicos entre 371-0001 y 371-9999 ambos inclusive. Hay 9.999 números dentro del intervalo. El número que indica el intervalo es 371-0001; el último es 371-9999.

EJERCICIOS Y CUESTIONES

1. ¿Qué hace la función incorporada INT?
2. ¿Qué hace la función incorporada RND?
3. Cuando las sentencias

```
40 LET U = INT (4.1)
50 LET V = INT (4.5)
60 LET W = INT (4.6)
70 LET X = INT (-9.1)
80 LET Y = INT (-9.5)
90 LET Z = INT (-9.9)
```

   se ejecutan, ¿cuáles son los valores asignados a U, V, W, X, Y y Z?
4. Escribir una sentencia en BASIC que redondee el valor de Q; es decir si su valor es 4.53, deberá redondear a 5; si su valor es 4.48 deberá redondear a 4, etc.
5. Escribir la sentencia en BASIC que redondee T dos lugares; es decir, si su valor es 2.68873, lo redondeará a 2.69; si su valor es 2.68453, deberá redondearlo a 2.68, etc.
6. Escribir una sentencia IF que consulte cuándo V es impar o par.
7. Escribir una sentencia o una serie de sentencias que dé el número resto entero cuando el entero V es dividido por el entero B. Por ejemplo si V es 17 y B es 5, el resto calculado es dos.
8. Verdadero o falso. la función RND puede dar estos números aleatorios

   .672, 1.674, 0.367, .9987

9. Verdadero o falso. La función RND puede dar estos números aleatorios

   .123, 0.0, .99999, 1.0000

10. Verdadero o falso. Cuando su programa utiliza números aleatorios, el nombre X deberá usarse únicamente en la función RND.

11. ¿Cómo debería interrogar Vd. al ordenador para que le dé una serie de números que no deberán repetirse cada vez que funcione el programa?
12. ¿Cómo le pediría al ordenador para que le dé a Vd. un solo número aleatorio comprendido entre 17 y 77 ambos inclusive?
13. Escribir un programa que dé la media de los primeros 100 números aleatorios de una serie no repetible de números aleatorios.
14. Escribir un programa que dé 1000 números aleatorios, pero imprima solamente cada 100 números; es decir, el programa tiene que imprimir el 100 avo, 200 avo... 1.000 avo número.
15. Escribir un programa que engendre números aleatorios enteros entre 1 y 100 inclusives e interroga a cada uno para saber si es igual a 53. Cuando el número aleatorio es igual a 53, el programa dice qué prueba (la 1.ª, 2.ª, 3.ª, etc.) dio aquel número.
16. Escribir un programa que simule la selección aleatoria de 100 bolas blancas o negras de una caja llena con igual número de bolas blancas que negras. Entonces el programa tiene que imprimir el número de bolas blancas seleccionadas y el número de bolas negras. Supongamos que después de cada elección, la bola elegida es situada fuera de la caja y que las bolas son mezcladas entre sí antes de que efectúe la selección.

CAPITULO **15**

Sentencias INPUT y RESTORE

Los datos de entrada para un programa en BASIC se proporcionan usualmente por la sentencia DATA. Cuando el dato se lee, puede examinarse, acumularse, imprimirse o utilizarse de otras muchas formas.

En ejemplos anteriores se ha señalado que los datos eran leídos solamente una vez. Los datos pueden sin embargo ser leídos muchas veces.

He aquí un ejemplo:

```
100 DATA 4, 5, 7, 3, 2, 1, 3, 8, 4, 6, 7, 9
110 READ A, B, C, D
120 PRINT A, B, C, D
130 READ E, F, G
140 PRINT E, F, G
150 RESTORE
160 READ H, K
170 PRINT H, K
180 READ L, M, N, O, P
190 PRINT L, M, N, O, P
200 END
```

La primera sentencia READ asigna 4, 5, 7 y 3 a A, B, C y D, respectivamente. Después de imprimir estos valores, el programa ejecuta la siguiente sentencia READ, que asigna 3, 2 y 1 a E, F y G respectivamente. Después de imprimir estos valores el programa ejecuta el comando RESTORE. Un “puntero” se mueve hacia atrás al comienzo de la sentencia DATA. La siguiente READ obtiene los valores 4 y 5 y los asigna a H y K. La sentencia final READ obtiene los valores 7, 3, 2, 1 y 3 y los asigna a L, M, N, O y P.

El comando RESTORE permite realizar la lectura de datos tantas veces como se desee.

El último programa no hizo gran cosa. Consideremos un ejemplo más práctico. Supongamos que tenemos estas sentencias DATA:

```
100 DATA 14563, 94, 14605, 16, 14712, 107, 15539, 88
110 DATA 16514, 27, 16630, 3, 17021, 211, 17524, 9
120 DATA 17536, 45, 17683, 7
```

La sentencia DATA contiene 10 conjuntos de valores. Cada conjunto contiene dos números. El primero en cada conjunto representa un número de pieza, y el segundo representa una cantidad en existencia. Los números en cada conjunto están relacionados. Esto significa que hay 94 unidades de existencia de la pieza 14563, 16 unidades de la pieza 14605, etc.

Supongamos que deseamos escribir un programa que acepte números de pieza introducidos desde el teclado del terminal, que explore la sentencia DATA hasta que es hallado el número de la pieza, en cuyo caso imprime ambos, el número de la pieza y la cantidad en existencia.

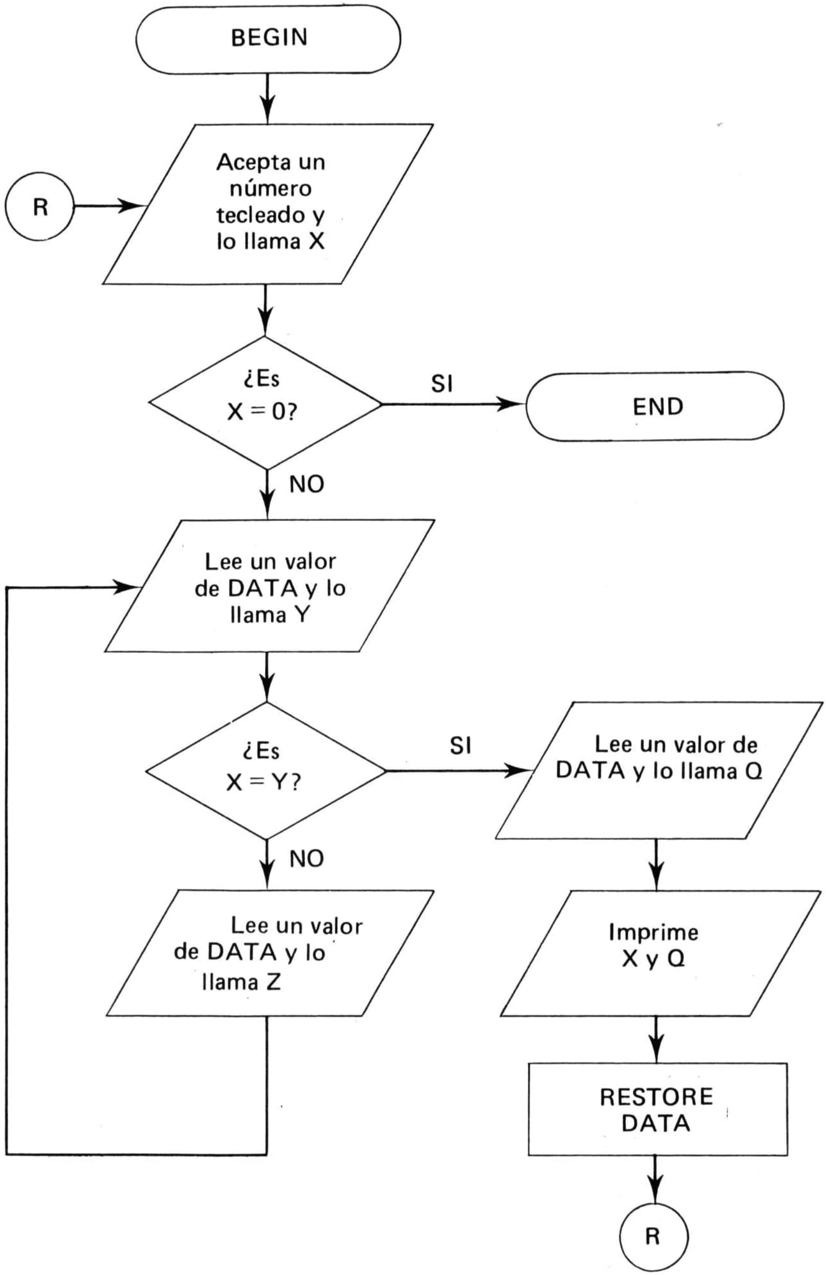
Un diagrama de flujo indicando la solución se halla en la página siguiente. He aquí el programa correspondiente:

```
100 DATA 14563, 94, 14605, 16, 14712, 107, 15539, 88
110 DATA 16514, 27, 16630, 3, 17021, 211, 17524, 9
120 DATA 17536, 45, 17683, 7
130 INPUT X
140 IF X = 0 THEN 230
150 READ Y
160 IF X = Y THEN 190
170 READ Z
180 GO TO 150
190 READ Q
200 PRINT X, Q
210 RESTORE
220 GO TO 130
230 END
```

El programa muestra la sentencia INPUT en este texto por primera vez. Cuando el ordenador ejecuta INPUT escribe una marca de interrogación sobre el papel del terminal. El ordenador está pidiendo que Vd. escriba un valor. En el ejemplo, aquel valor se le asigna a X. El valor representa un número de pieza.

Habiendo aceptado un valor de X, el programa comprueba si Vd. ha escrito cero. Si lo hizo el programa termina. Si no el programa le lista DATA hasta que encuentre el valor que Vd. dio.

Habiendo encontrado el número de la pieza que está siendo explorado, el programa lee el número siguiente, Q. Este valor representa la cantidad en existencia de la pieza. Entonces el programa imprime el núme-


BEGIN
Acepta un número tecleado y lo llama X
R
¿Es X = 0?
SI
END
NO
Lee un valor de DATA y lo llama Y
¿Es X = Y?
SI
Lee un valor de DATA y lo llama Q
NO
Lee un valor de DATA y lo llama Z
Imprime X y Q
RESTORE DATA
R

ro de la pieza y la cantidad en existencia. A continuación, el programa restaura los datos de la sentencia DATA y va hacia atrás a la sentencia INPUT para aceptar otro número de pieza para la exploración. La exploración para la nueva pieza se inicia al comienzo de la sentencia DATA.

En el programa, cuando X no es igual a Y, abandona la cantidad en existencia de Y para leer el siguiente valor, lo llama Z. Z no se usa nunca.

Cuando Vd. escribe RUN, el programa da esta salida:

```
?14605
 14605        16
?17021
 17021        211
?0
READY
```

El programa imprime toda la salida señalada en la figura excepto para los números de pieza que siguen al signo de interrogación. Estos fueron escritos por el usuario.

La sentencia INPUT puede referirse a más de un valor. Supongamos que la sentencia INPUT es la

```
500 INPUT A, B, C
```

Cuando el ordenador imprime el signo de interrogación, el usuario deberá escribir tres valores separados por comas. Si él escribe menos o más que tres, el programa imprimirá automáticamente

```
INPUT IN WRONG FORMAT–PLEASE RETYPE
```

El usuario puede escribir entonces los tres números que le son exigidos.

Como Vd. ha podido ver, la sentencia INPUT permite la interacción entre el ordenador y el programador. El ordenador escribe un mensaje y el usuario responde; entonces el ordenador escribe un mensaje otra vez y el usuario responde de nuevo; la conversación continúa durando todo lo que sea necesario para resolver el problema.

He aquí un segmento de un programa para enseñar matemáticas que un programador podría crear:

```
10  PRINT "HOLA, VAMOS A APRENDER MATEMATICAS."
20  PRINT "¿DIGAME CUANTO ES 6 POR 8?"
30  INPUT A
40  IF A = 48 THEN 70
50  PRINT "NO, NO ES CORRECTO; INTENTELO DE NUEVO."
60  GO TO 30
70  PRINT "MUY BIEN."
80  PRINT "¿CUANTO VALE EL SENO DE 30 GRADOS?"
90  INPUT S
100 IF S = .5 THEN 130
110 PRINT "NO, NO ES CORRECTO; INTENTELO DE NUEVO."
120 GO TO 90
130 PRINT " ¡CORRECTO!"
140 LET C = 1
150 PRINT "AHORA, ¿CUANTO ES LA RAIZ CUADRADA DE 144?
160 INPUT R
170 IF R = 12 THEN 240
180 LET C = C + 1
190 IF C > 3 THEN 220
200 PRINT "NO ES CORRECTO. INTENTELO DE NUEVO."
210 GO TO 160
220 PRINT "POR FAVOR, CONSULTELO A SU PROFESOR."
230 GO TO 2000
240 PRINT " ¡BIEN!
        .
        . sentencias adicionales no escritas
        .
        .
        .
2000 END
```

El programa puede funcionar durante horas enseñando matemáticas, ciencias o inglés. Observemos que al comienzo de la línea 140, el programa da al estudiante tres posibilidades para responder una cuestión concerniente a la raíz cuadrada de 144. Es una buena idea poner en un programa de enseñanza un contador de las respuestas incorrectas de forma que el estudiante no pueda dar demasiadas respuestas erróneas, así se gana tiempo de ordenador.

EJERCICIOS Y CUESTIONES

1. ¿Cuál es la función de la sentencia RESTORE?
2. En ausencia de RESTORE, ¿cuántas veces pueden ser leídos los valores en una sentencia DATA?

3. ¿Qué mensaje imprimirá el ordenador cuando se hace un intento de lectura de una sentencia DATA que no tiene más valores disponibles?
4. Cuando el ordenador ve una sentencia INPUT, ¿qué escribe sobre el papel de la terminal de salida?
5. Cuántos valores espera el ordenador en respuesta a

```
50 INPUT L, F, V
```

6. Estudiar este programa:

```
100 PRINT "ESCRIBA UN NUMERO ENTERO, POR FAVOR."
110 INPUT N
120 IF N = 5 THEN 150
130 PRINT "NO ES ESTE."
140 GO TO 100
150 PRINT " ¡SI, ES EL NUMERO!"
160 END
```

Cuando el usuario escribe RUN, ¿qué mensaje escribe el ordenador?

7. En el programa anterior, ¿qué respuesta debería proporcionar el usuario como contestación a la exigencia del ordenador?
8. ¿Qué hace el ordenador con el número que escribe el usuario?
9. Si el usuario escribe 5 ¿qué hace el ordenador?
10. Si el usuario no escribe 5, ¿qué hace el ordenador?
11. Estudiar este programa de "juego":

```
100 LET R = INT (RND (X) * 100) + 1
110 PRINT "TENGO UN NUMERO."
120 PRINT "¿PUEDE VD. ACERTARLO?"
130 PRINT "TECLEE SU INTENTO."
140 INPUT G
150 IF R = G THEN 220
160 IF R > G THEN 200
170 PRINT "SU NUMERO ES DEMASIADO ALTO."
180 PRINT "INTENTELO DE NUEVO."
190 GO TO 140
200 PRINT "SU NUMERO ES DEMASIADO BAJO."
210 GO TO 180
220 PRINT " ¡ENHORABUENA!"
240 PRINT "TECLEE 1 SI SI Y 0 SI NO."
250 INPUT A
260 IF A = 1 THEN 100
270 END
```

¿Qué mensaje imprimirá el ordenador cuando el programa comience?

12. En el programa anterior ¿qué debe esperar el ordenador que le dé Vd. cuando él escribe un signo de interrogación?
13. ¿Qué hará el ordenador con el número que Vd. escribe?
14. Si su conjetura es igual a R ¿qué imprimirá el ordenador?
15. Si su conjetura no es igual a R ¿qué hecho importante le dará el ordenador a usted?
16. ¿Cómo le preguntará el programa si Vd. desea continuar con el juego?
17. Escribir un programa que enseñe a un estudiante cuáles son los productos de 6 x 8 y 7 x 9. Usar la sentencia INPUT para obtener una respuestas del usuario.
18. Escribir un programa que lea los siguientes datos dos veces:

```
100 DATA 18, 92, 47, 36, 62, 999
```

El valor 999 es ficticio e indica el fin de los datos.

En el primer tiempo en que se leen los datos, el programa multiplica cada valor por dos y a continuación lo imprime; la segunda vez en que el dato es leído, el programa multiplica por tres cada valor y lo imprime.

CAPITULO **16**

Matrices y subíndices

Hay veces en que usar datos directamente de una sentencia DATA con el fin de resolver un problema es extremadamente difícil si no imposible. Es verdad que usando RESTORE uno puede tener los valores en una sentencia DATA y leerlos una y otra vez. Sin embargo, hay veces en que sería mejor si todos los valores data en una sentencia DATA fueran transferidos a un área de trabajo de la memoria del ordenador y el programa entonces trabajará directamente con dicha área.

Con el fin de aprender cómo transferir valores de una sentencia DATA y cómo trabajar con los valores una vez que la transferencia ha sido realizada, deberemos primero aprender cómo usar la sentencia DIM.

La sentencia DIM toma estas formas:

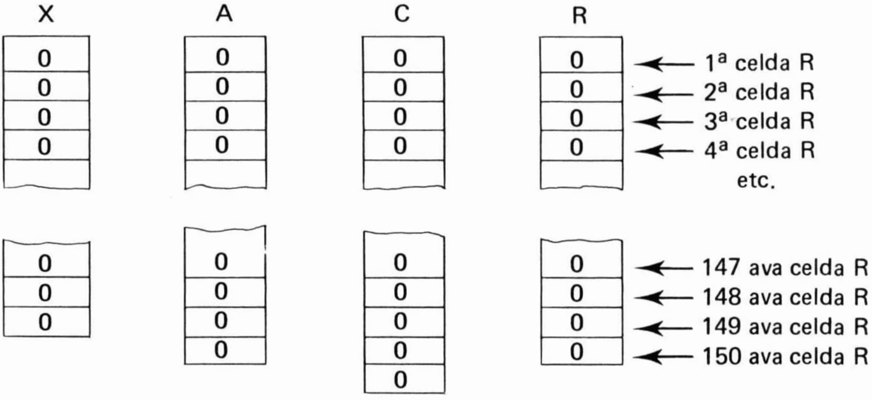
```
200 DIM X (50)

210 DIM A (200), C (300), R (150)
```

Con ellas estamos dirigiendo al ordenador a que reserve 50 celdas de la memoria llamadas todas X. Además, le estamos diciendo al ordenador que haga lo mismo con 200 celdas llamadas A, con 300 celdas llamadas C y con 150 celdas llamadas R. (Se pueden dar varias sentencias DIM en un programa.) Por conveniencia Vd. puede pensar las celdas de la memoria en el ejemplo anterior como si estuvieran ordenadas de la forma que se muestra en la página siguiente.

La ilustración indica que cuando las celdas de la memoria son reservadas, ellas reciben un cero como valor inicial.

Puesto que todas las X son iguales y tienen el mismo nombre, deberemos sistematizar un método para distinguir unas de otras. (Tenemos el mismo problema con las celdas A, C y R.) Aquí y donde los *subíndices*



entran en el cuadro. Un subíndice es un número entero, o el resultado de un cálculo o un nombre que representa un número entero. Cuando se sitúan entre paréntesis, los números subíndices nos dicen en qué X, A, C ó R estamos interesados. Aquí por ejemplo, hay algunas sentencias que emplean subíndices:

```
100 DIM F (15)

110 LET F (3) = 4.9

120 LET F (14) = 3.34

130 LET F (15) = F (3) + 8

     .
     .
     .
     .
     .
     .
     .
```

Este segmento de programa reserva primero 15 celdas de memoria llamadas F. Entonces asigna el valor 4.9 a la tercera posición de memoria F, el valor 3.34 a la catorceava y el valor 12.9 a la quinceava.

Después de que estas instrucciones han sido ejecutadas, los valores de F aparecen así:

F
0
0
4.9
0
0
0
0
0
0
0
0
0
0
3.34
12.9

El grupo de valores señalados anteriormente se dice que es un vector. Hay 15 valores en los vectores y el nombre del vector es F. Un vector queda establecido mediante el uso de la sentencia DIM. (Véase línea 100 en el ejemplo.) La sentencia DIM que reserva un vector puede colocarse en cualquier parte de un programa en BASIC, pero deberá preceder al uso que el programa haga del vector.

Los números señalados entre paréntesis son ejemplos de subíndices. Los subíndices apuntan a celdas particulares de un vector. Los subíndices no pueden nunca ser cero ni números negativos; ellos no pueden nunca ser mayores que el tamaño del vector. En el ejemplo los subíndices pueden variar de 1 a 15.

Los vectores pueden contener miles de celdas. Los tamaños máximos permitidos varían en los diversos sistemas.

Vd. está siempre en lo correcto si opera con un tamaño de 1000 ó menos. También esta en lo correcto si considera el tamaño máximo del vector y actualmente usa menos celdas que el máximo.

Vamos a transferir algunos valores de una sentencia DATA a un vector:

```
100 DATA 3, 16, 14, 6, 8, 7, 9, 18, 21, 1, 5, 12

110 DIM P (12)

120 FOR K = 1 TO 12
```

```
130 READ P (K)
140 NEXT K
        .
        .
        .
        .
```

Este segmento de programa lee 12 valores de la sentencia DATA y asigna los valores a P. Observamos que el subíndice usado en la línea 130 no es un número entero actual, sino más bien la representación nominal del nombre. El valor asignado a K varía de uno en uno mientras el valor actual de K alcanza 12 y es usado como un subíndice con aquel valor.

He aquí el aspecto del vector P antes de que el bucle FOR/NEXT haya sido ejecutado:

P
0
0
0
0
0
0
0
0
0
0
0
0

Y he aquí cómo aparece el vector después de que el bucle FOR/NEXT ha sido ejecutado:

P
3
16
14
6
8
7
9
18
21
1
5
12

Habiendo situado estos 12 valores en un vector, podemos dirigirnos al programa para trabajar con ellos. Suponemos que la siguiente parte del programa ejemplo es ésta:

```
      .
      .
      .
      .
      .
150 FOR N = 1 TO 12
160 PRINT N, P (N)
170 NEXT N
```

El programa imprimirá 12 líneas. Cada línea contendrá un número variando de 1 a 12 inclusives y uno de los 12 valores de P. Vd. verá esta salida:

1	3			
2	16			
3	14			
4	6			
5	8			
.	.			
.	.			
.	.			
.	.			

Supongamos ahora que los 12 valores deben imprimirse en orden inverso. Podemos añadir estas sentencias:

```
      .
      .
      .
      .
180 FOR L = 1 TO 12
190 PRINT P (13 – L)
200 NEXT L
```

El programa calcula el primer subíndice de la expresión 13 – L. Puesto que L contiene el valor 1 cuando el FOR/NEXT bucle es ejecutado por primera vez, 13 – L da el valor 12. En consecuencia, la posición de la doceava celda del vector P se imprime. Entonces L cambia a 2, 13 – L vale 11, y el ordenador imprime la 11.ª posición de la celda de P.

La última vez en que el FOR/NEXT se ejecuta, el valor asignado a L es 12. La expresión 13 – L da 1 y el ordenador imprime el primer valor del vector P.

Los ejemplos dados anteriormente han mostrado que los subíndices pueden ser números enteros constantes.

Ejemplo:

```
2000 LET D = P (3)

2100 PRINT P (11)
```

o bien que los subíndices pueden ser nombres:

Ejemplo:

```
2000 PRINT P (L)
2300 LET F = F + P (N)
```

o bien que los subíndices pueden ser resultados enteros de cálculos.

Ejemplo:

```
3000 LET M = W + P (13 – J)
3010 PRINT P (13 – N)
```

Es importante observar que un vector no siempre debe ser accesible por el mismo subíndice. En los ejemplos anteriores, el vector P era accesible por los subíndices llamados K, N y L entre otros. Sin embargo es totalmente correcto para un programador el emplear siempre el mismo subíndice. Una sentencia FOR siempre reinicializa un subíndice al valor de comienzo señalado directamente en la sentencia.

He aquí un problema que en sí mismo no es importante, pero tiene un valor educacional porque muchos programas usan la técnica en la solución de problemas más complicados. Supongamos que tenemos esta sentencia DATA:

```
10 DATA 18, 9, 7, 14, 6, 4, 1, 17, 21, 16, 2, 12, 3, 11, 5
```

Queremos escribir un programa que transfiera estos 15 valores a la memoria del ordenador. A continuación el ordenador examinará los valores uno por uno, intercambiando valores hasta que el mayor descienda al fondo del vector. He aquí un programa que realiza esta tarea:

```
10 DATA 18, 9, 7, 14, 6, 4, 1, 17, 21, 16, 2, 12, 3, 11, 5

20 DIM W (15)

22 FOR J = 1 TO 15

24 READ W (J)

26 NEXT J

30 FOR K = 1 TO 14

40 IF W (K + 1) >= W (K) THEN 80

50 LET T = W (K)

60 LET W (K) = W (K + 1)

70 LET W (K + 1) = T

80 NEXT K

90 PRINT W (15)

100 END
```

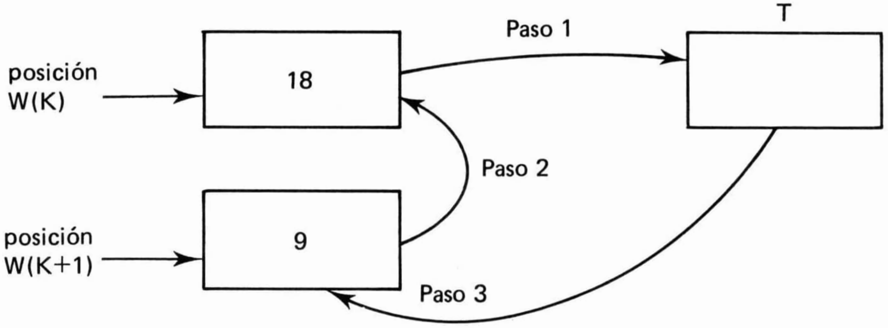
Este programa examina 14 pares de números. (K varía desde 1 hasta 14.) El primer par de números comprende los valores

18 [18] W (K) (la primera celda del vector)

y

9 [9] W (K + 1) (la segunda celda del vector)

Si W (K + 1) es mayor que o igual a W (K), los números están en secuencia y no tienen que ser tocados. Si el valor de W (K + 1) no es mayor o igual que W (K), entonces W (K) será mayor que W (K + 1). Los dos números no están en secuencia y los dos valores deben intercambiarse. Tres secuencias en BASIC se necesitan para intercambiar los valores. El ejemplo muestra que cuando se necesita un intercambio, el valor de W (K) se almacena temporalmente en una celda llamada T; el valor de W (K + 1) se le asigna a W (K); y finalmente el valor almacenado en T se asigna a W (K + 1). Se puede visualizar el procedimiento mirando el diagrama siguiente:



Si se usan solamente dos sentencias para hacer el intercambio, uno de los dos valores sería destruido. Por ejemplo,

```
50 LET W (K) = W (K + 1)

60 LET W (K + 1) = W (K)
```

pone ambos valores con el de W (K + 1).

En el ejemplo, cuando el valor de K es aumentado en 1, un nuevo par de valores es examinado. Cuando el valor de K alcanza 14, el par de valores considerados son

W (14)

y

W (15)

El bucle se determina y el mayor de los 15 números es encontrado definitivamente en la posición 15.ª de la celda de W.

En este capítulo hemos discutido meramente la mecánica de manejar vectores y de acceder a ellos usando subíndices. Discutiremos algunos ejemplos prácticos más que fueron dados en el texto. En el capítulo siguiente veremos cómo pueden usarse los vectores para clasificar valores o para explorar tablas.

EJERCICIOS Y CUESTIONES

1. ¿Cuál es la función de la sentencia DIM?
2. ¿Qué es un vector?
3. Cuando se define un vector empleando una sentencia DIM, ¿qué dos características debe dar concernientes al vector?
4. ¿Qué valores iniciales espera encontrar en las celdas de la memoria de un vector?
5. Verdadero o falso. Un vector puede definirse mayor que el que Vd. piensa que necesita, pero no menor.
6. Verdadero o falso. Uno puede resolver fácilmente todos los problemas usando DATA sin tener que utilizar vectores.
7. ¿Qué falta en este vector?

```
400 DIM W (500), F (25), (75)
```

8. ¿Cuál es el nombre de este vector?

```
500 DIM L (200)
```

9. ¿Cuál es el tamaño del vector en la cuestión 8?
10. ¿Cuál es la definición de subíndice?
11. Dadas las sentencias DATA y DIM señaladas a continuación, escribir tres sentencias que lean los datos en el vector N.

```
10 DIM N (10)
20 DATA 14, 16, 15, 11, 19, 17, 6, 8, 21, 43
```

12. Dada la sentencia DIM señalada a continuación, escribir una sentencia que sitúe el valor 9.5 en la quinta posición de R.

```
100 DIM R (80)
```

13. Dar las tres formas que pueden tomar los subíndices.
14. ¿Pueden ser negativos los subíndices?
15. ¿Los subíndices deberán siempre ser números enteros?

16. Verdadero o falso. Los subíndices no pueden nunca ser cero ni mayores que los tamaños dados a los vectores a los que se aplican.
17. Usando esta sentencia DIM:

```
200 DIM E (100)
```

escribir tres sentencias que almacenen los valores 1 hasta 100 en celdas consecutivas del vector.

18. Estudiar estas sentencias:

```
10 DIM T (10), V (10), X (10)
20 DATA 9, 7, 5, 11, 6, 8, 2, 8, 17, 10
```

escribir cuatro sentencias que almacenen estos valores en el vector T y a contición lo copien en el vector V.

19. Seguir la instrucción dada en la cuestión 18 anterior, pero escribir tres sentencias más que copien el vector V en el vector X pero en orden inverso.
20. Continuando la asignación dada anteriormente. Escribir un conjunto de sentencias que encuentre el menor valor en el vector X y lo imprima.
21. Continuando la asignación dada anteriormente. Escribir un conjunto de sentencias que halle el mayor valor en el vector X, imprima este valor, y diga también su posición en el vector.

CAPITULO **17**

Uso de tablas o vectores

Muchos problemas con frecuencia implican una clasificación u ordenación de números. Los vectores pueden usarse efectivamente como una ayuda para clasificar una colección de números.

Existen muchas técnicas que pueden usarse en una clasificación u ordenación. El *buble Sort* (clasificación por el método de la "burbuja") es una de las más difíciles de entender y usar. Usemos este método para clasificar en sucesión creciente los 12 valores en esta sentencia DATA. (Supongamos siempre una sucesión creciente):

```
100 DATA 44, 24, 18, 56, 6, 78, 21, 27, 18, 34, 15, 31
```

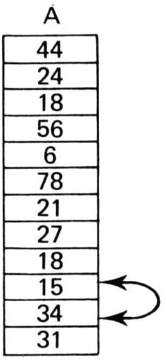
El primer objetivo del programa de selección es situar estos valores en un vector A de 12 celdas:

A
44
24
18
56
6
78
21
27
18
34
15
31

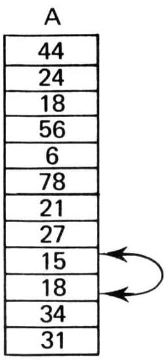
Ahora el programa examina 11 parejas de números comenzando en la parte inferior del vector. Si un par de números está ya en sucesión ascendente, el programa deja a los números tal y como están; si los números no están en secuencia, el programa intercambia los dos valores. En el

ejemplo, el primer par de números examinados, 15 y 31, están ya en sucesión creciente, por consiguiente el programa los deja igual.

El siguiente par de números para ser examinados contiene los valores 34 y 15. Puesto que ellos no están en secuencia, el programa intercambia los dos valores:



El paso siguiente es consultar el par siguiente, 18 y 15. Puesto que ambos no están en secuencia, el programa los intercambia. El vector aparece ahora así:



De igual forma el programa trabaja hacia la parte superior del vector. Habiendo terminado este procedimiento, el programa ha situado el menor número del vector en la parte superior de éste. Se dice que el valor 6 ha flotado como una burbuja hasta la cabecera del vector.

El vector aparece así:

A
6
44
24
18
56
15
78
21
27
18
34
31

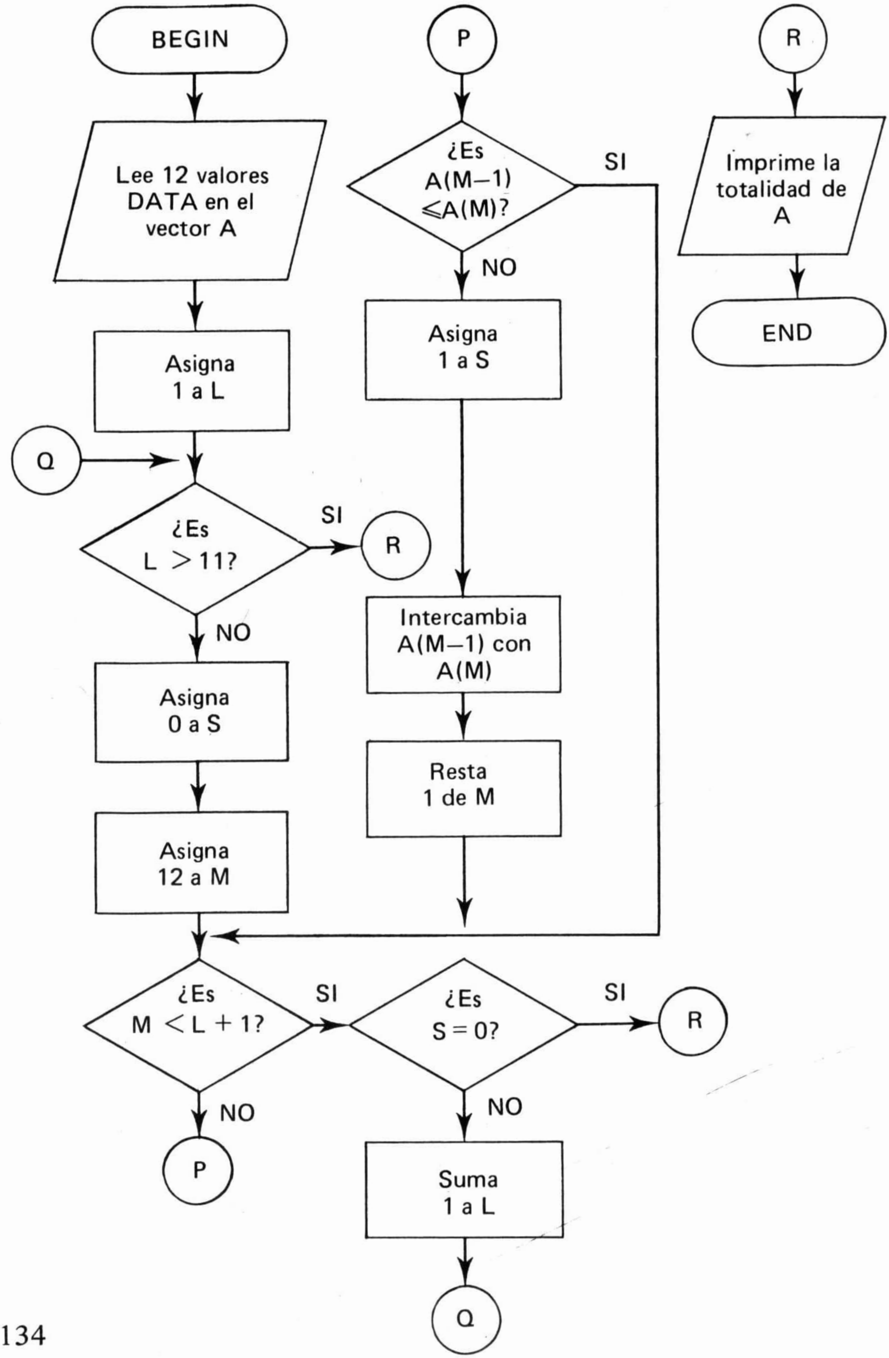
Por supuesto, los valores no han sido todavía ordenados por completo. Se necesita un paso segundo. El programa no debe volver a considerar la celda de la cabecera del vector puesto que se sabe incuestionablemente que la celda de la parte superior del vector contiene el menor valor. Cuando el segundo paso a través del vector ha finalizado, el segundo número menor está situado en la segunda celda del vector a partir de su parte superior. El vector aparece como éste:

A
6
15
44
24
18
56
18
78
21
27
31
34

El segundo paso no es el último. Puesto que existen 12 números en el vector, un máximo de 11 pasos pueden necesitarse. Cada paso será más corto que el que le precede. El último paso necesita examinar solamente los dos números del fondo del vector. El diagrama de flujo que aparece en la página siguiente indica cómo el programa ha de funcionar.

He aquí el programa:

```
100 DATA 44, 24, 18, 56, 6, 78, 21, 27, 18, 34, 15, 31
110 DIM A (12)
```


BEGIN
Lee 12 valores DATA en el vector A
Asigna 1 a L
Q
¿Es L > 11?
SI
R
NO
Asigna 0 a S
Asigna 12 a M
¿Es M < L + 1?
SI
NO
P
¿Es S = 0?
SI
R
NO
Suma 1 a L
Q
P
¿Es A(M–1) ≤A(M)?
SI
NO
Asigna 1 a S
Intercambia A(M–1) con A(M)
Resta 1 de M
R
Imprime la totalidad de A
END

```
120 FOR K = 1 TO 12
130 READ A (K)
140 NEXT K
150 FOR L = 1 TO 11
160 LET S = 0
170 FOR M = 12 TO L + 1 STEP -1
180 IF A (M + 1) <= A (M) THEN 230
190 LET S = 1
200 LET T = A (M - 1)
210 LET A (M - 1) = A (M)
220 LET A (M) = T
230 NEXT M
240 IF S = 0 THEN 260
250 NEXT L
260 FOR N = 1 TO 12
270 PRINT A (N)
280 NEXT N
290 END
```

En este programa, L cuenta el número máximo de pasos que deberá hacer el programa para ordenar los 12 números. Este máximo es 11. El valor de M controla la longitud de cada paso. El primer paso accede valores de A (12) a A (2). Véase la sentencia 170 y advirtamos que L + 1 es igual a 2 durante el primer paso. Cada valor accedido es comparado con el que le precede inmediatamente. Véase la sentencia 210: el subíndice M permite el examen de dos valores consecutivos. El segundo paso procesa valores de A de A (12) a A (3). (El valor de L + 1 es igual a 3.)

El valor de S se consulta para la terminación de la exploración. Después de la terminación de un paso, si el valor de S es 0, el programa termina inmediatamente puesto que ningún cambio de valores puede hacerse durante el paso. Examinemos la línea 190. Vd. verá que el valor S cambia de 0 a 1 cuando y sólo cuando se hace un intercambio de valores durante un paso.

El resto del programa es autoexplicativo. Observemos que las líneas 120 hasta 140 leen los valores de los datos a clasificar del vector A mientras las líneas 260 hasta 280 imprimen los valores clasificados.

En BASIC se pueden considerar hasta matrices de dos dimensiones. Esta tarea se realiza con la sentencia DIM, por ejemplo:

```
100 DIM B (4, 5)
```

El 4 en la sentencia se refiere al número de filas en la matriz y el 5 se refiere al número de columnas. Se puede representar esta matriz como sigue:

B

0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0

La matriz B tiene
4 filas y
5 columnas

se pueden colocar valores en la matriz bidimensional de esta forma:

```
100 DIM B (4, 5)

110 DATA 3, 7, 8, 4, 0, 5, 6, 8, 4, 9, 7

120 DATA 6, 6, 0, 1, 8, 9, 3, 2, 4

130 FOR K = 1 TO 4

140 FOR N = TO 5

150 READ B (K, N)

160 NEXT N

170 NEXT K
```

Tenemos un bucle dentro de un bucle. El valor K se inicializa a 1, y los valores N varían entonces de 1 a 5 ambos inclusive. Luego K vale 2 y se mantiene fijo mientras los valores de N varían de 1 a 5 nuevamente. El procedimiento se repite con K igual a 3 y manteniéndose en este valor, etc. Los valores de K y N varían paso a paso de esta forma:

Fila (K)	*Columna (N)*
1	1
1	2
1	3

Fila (K)	*Columna (N)*
1	4
1	5
2	1
2	2
2	3
2	4
2	5
3	1
3	2
3	3
3	4
3	5
4	1
4	2
4	3
4	4
4	5

Puesto que K y N proporcionan los subíndices de fila y columna para la matriz B, los 20 valores leídos de la sentencia DATA se almacenan en B de esta forma:

B

3	7	8	4	0
5	6	8	4	9
7	6	6	0	1
8	9	3	2	4

Ahora veamos lo que sucedería si la sentencia en la línea 150 fuera cambiada de

```
150  READ B (K, N)
```

a

```
150  READ B (N, K)
```

En la última sentencia, N representa filas y K representa columnas. Los valores de los subíndices K y N se producen todavía como antes.

En consecuencia los 20 valores en la sentencia DATA se almacenan en la matriz B como sigue:

B

3	0	4	6	9
7	5	9	0	3
8	6	7	1	2
4	8	6	8	4

Veamos una aplicación práctica de una matriz o tabla de dos dimensiones. Preparemos primero esta tabla:

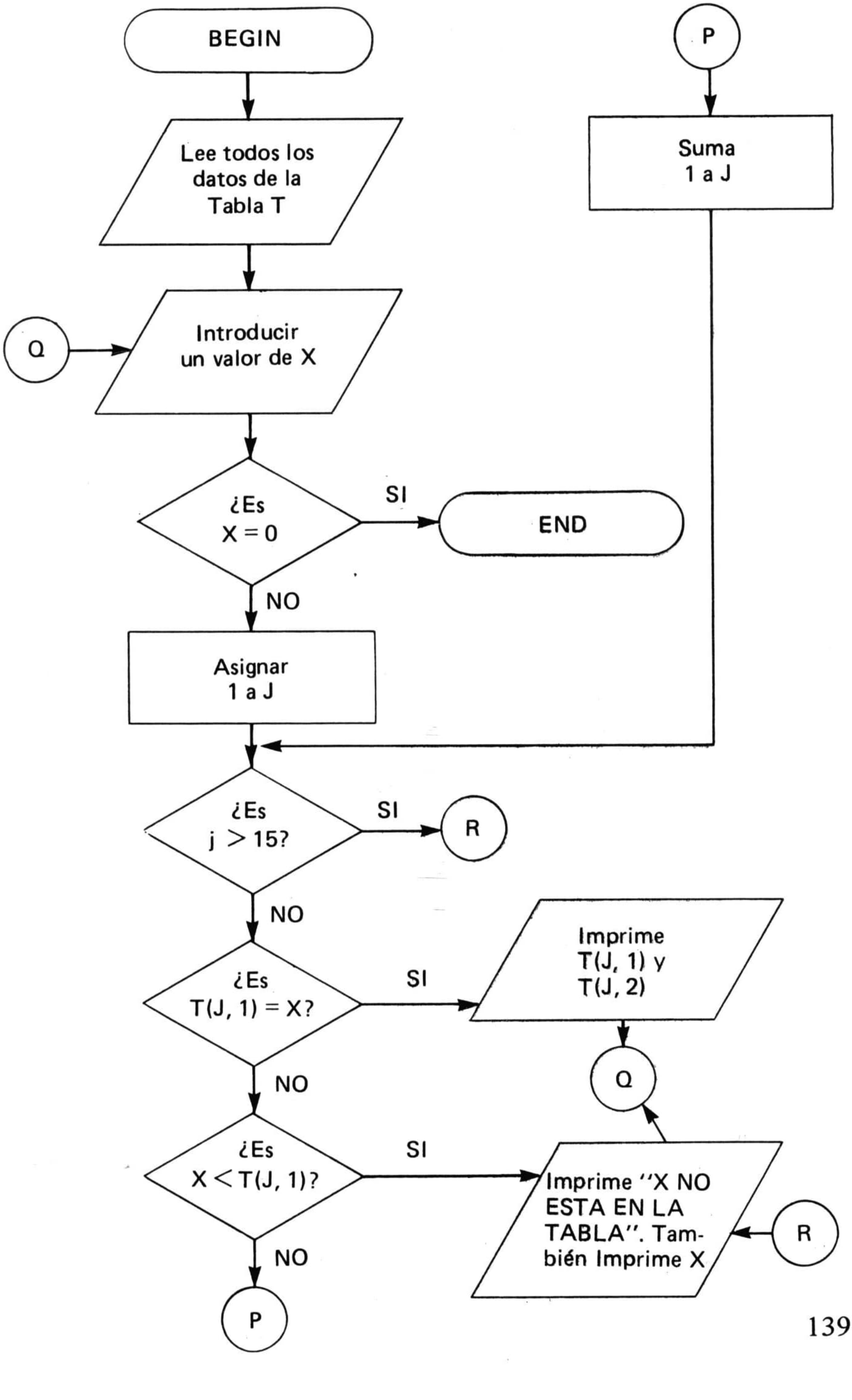
T

246	6
255	18
261	21
275	0
283	45
294	17
298	19
306	4
315	163
326	17
335	3
343	81
356	46
364	18
375	16

Los valores en la columna de la izquierda representan números de pieza; los valores en la columna de la derecha representan cantidades de existencia. Escribiremos un programa que obtiene un número de pieza del teclado, lo halla en la columna de la izquierda de la tabla T e imprime ambos el número de la pieza y la cantidad en existencia. El diagrama de flujo para este programa se halla en la página siguiente.

He aquí el programa:

```
100 DATA 246, 6, 255, 18, 261, 21, 275, 0, 283, 45
110 DATA 294, 17, 298, 19, 306, 4, 315, 163, 326, 17
```


BEGIN
Lee todos los datos de la Tabla T
Q
Introducir un valor de X
¿Es X = 0
SI
END
NO
Asignar 1 a J
¿Es j > 15?
SI
R
NO
¿Es T(J, 1) = X?
SI
Imprime T(J, 1) y T(J, 2)
Q
NO
¿Es X < T(J, 1)?
SI
Imprime "X NO ESTA EN LA TABLA". También Imprime X
R
NO
P
P
Suma 1 a J

```
120 DATA 335, 3, 343, 81, 356, 46, 364, 18, 375, 16
130 DIM T (15, 2)
140 FOR J = 1 TO 15
150 FOR K = 1 TO 2
160 READ T (J, K)
170 NEXT K
180 NEXT J
190 INPUT X
200 IF X = 0 THEN 290
210 FOR J = TO 15
220 IF T (J, 1) = X THEN 270
230 IF X < T (J, 1) THEN 250
240 NEXT J
250 PRINT "X NO ESTA EN LA TABLA", X
260 GO TO 190
270 PRINT T (J, 1), T (J, 2)
280 GO TO 190
290 END
```

Observemos unos pocos puntos sobre este programa. Primero advirtamos que la tabla T es un array 15 por 2. Estos significa que él tiene 15 filas y 2 columnas. Es posible explorar solamente la primera columna (la única que mantiene los números de pieza), fijar el subíndice de la columna en 1 mientras varían los subíndices de fila de 1 en 1 hasta 15. En el ejemplo, J es el subíndice que controla la fila.

En la línea 230 el programa comprueba para ver cuándo serán inútiles exploraciones posteriores en la tabla. Observemos que los números de pieza se ordenan en una sucesión numérica creciente. Cuando X, el número de pieza que nosotros estamos explorando, es menor que el número de pieza que está siendo examinado, en la tabla, la exploración del resto de la tabla es inútil y la exploración puede detenerse. El programa imprime un mensaje advirtiendo que el número de pieza que está siendo explorado no está en la tabla.

Entonces se da al usuario otra oportunidad para teclear correctamente el número de pieza.

Adviértase que se imprime el mismo mensaje si se realizan completamente el bucle de exploración entre las líneas 210 y 240 y el valor de X no se encuentra.

El usuario le dice al programa que cese de funcionar escribiendo 0 cuando el programa pide otro valor de X.

Finalmente estudiamos la acción de los bucles READ que transfieren los valores que están en la sentencia DATA a la tabla T. El bucle exte-

rior que controla las filas varía de 1 a 2. Los subíndices así generados funcionan como sigue:

Fila	*Columna*
1	1
1	2
2	1
2	2
3	1
3	2
4	1
4	2
.	
.	
.	
.	
.	
14	1
14	2
15	1
15	2

Los valores de los datos han sido registrados en la sentencia DATA en el orden en que los subíndices los aceptarán, siendo luego almacenados los valores en la tabla T.

EJERCICIOS Y CUESTIONES

1. ¿Por qué el método de la burbuja tiene este nombre?
2. En el método de la burbuja, ¿cuál es el objetivo del primer paso completo en que los números a ser clasificados están en sucesión creciente?
3. En el método de la burbuja, ¿cuál es el número máximo de pasos necesarios antes de que todos los valores estén en sucesión creciente?
4. ¿Cómo un programa que clasifica usando el método de la burbuja determina cuándo la terminación del programa es necesaria?
5. Describir el método de la burbuja de forma que en lugar de que el menor valor vaya creciendo hacia la cabecera del vector durante cada paso, sea el mayor valor el que disminuya hacia el fondo del vector. Todos los otros detalles permanecen igual; las parejas de números se comparan, y los intercambios se hacen cuando los números no están en secuencia, se necesitan pasos múltiples, la terminación completa es posible, etc. Usar 11 valores en el vector.
6. ¿Qué es una tabla bidimensional?

7. ¿En la sentencia DIM siguiente cuáles son las filas y las columnas definidas pc la tabla G?

```
200 DIM G (10, 12)
```

8. Escribir las sentencias BASIC correctas para poner estos valores en la tabla H:

H

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20

No usar una sentencia DATA para resolver este problema.

9. Escribir las sentencias BASIC correctas para situar estos valores en la tabla L:

1	5	9	13
2	6	10	14
3	7	11	15
4	8	12	16

Usar la sentencia DATA en la solución de este problema.

10. Supongamos que ha sido creada esta tabla:

136	2.45
139	7.50
145	8.36
149	9.47
157	6.20
163	4.35
168	6.70
172	2.45

Escribir las sentencias correctas en BASIC que suman los valores en la columna de la derecha. Hacer que estas sentencias impriman la suma resultante.

CAPITULO **18**

Operaciones matriciales

Una matriz es una tabla de números rectangular o cuadrada. Típicamente se establece por una sentencia DIM el nombre de la matriz y su tamaño. Un bucle se emplea generalmente para cargar la matriz con números.

En BASIC, se dispone de varios tipos de sentencia para facilitar el trabajo con matrices. Como una introducción a las operaciones matriciales, consideremos las sentencias MAT READ y MAT PRINT. Estudiemos este programa ejemplo que no usa sentencias MAT:

```
100 DIM Q (5, 5)
110 FOR F = 1 TO 5
120 FOR C = 1 TO 5
130 READ Q (F, C)
140 NEXT C
150 NEXT F
160 FOR F = 1 TO 5
170 FOR C = 1 TO 5
180 PRINT Q (F, C)
190 NEXT C
200 NEXT F
210 DATA 3, 6, 7, 9, 15, 17, 18, 21, 24, 30
220 DATA 33, 40, 42, 43, 45, 47, 49, 51, 58, 63
230 DATA 65, 78, 84, 85, 87
240 END
```

El programa tiene un bucle dentro de otro bucle para leer valores en la matriz Q. En los bucles, F representa a las filas y C a las columnas. (Se podrían haber escogido otros nombres, en vez de F y C, aunque estos son más mnemotécnicos.) La matriz es cargada de la forma mostrada en la página siguiente.

La parte del programa que lee valores en la matriz hace variar C con mayor rapidez que F. Así, F vale 1 mientras C recorre el ciclo de 1 has-

Filas ↓ / Columnas →	1	2	3	4	5
1	3	6	7	9	15
2	17	18	21	24	30
3	33	40	42	43	45
4	47	49	51	58	63
5	65	78	84	85	87

ta 5; entonces F avanza a 2, y se mantiene, mientras C recorre el ciclo de 1 a 5 otra vez. Esto es lo que sucede hasta que la matriz Q se carga.

La siguiente parte del programa tiene un bucle interior a un bucle que imprime la matriz. Otra vez F significa la fila y C la columna.

Consideremos el mismo programa usando sentencias matriciales:

```
100 DIM Q (5, 5)

110 MAT READ Q

120 MAT PRINT Q

130 DATA 3, 6, 7, 9, 15, 17, 18, 21, 24, 30

140 DATA 33, 40, 42, 43, 45, 47, 49, 51, 58, 63

150 DATA 65, 78, 84, 85, 87

160 END
```

Vd. puede ver que MAT READ Q trabaja de la misma forma que

```
FOR F = 1 TO 5
FOR C = 1 TO 5
READ Q (F, C)
NEXT C
NEXT F
```

y MAT PRINT Q trabaja como

```
FOR F = 1 TO 5
FOR C = 1 TO 5
```

```
PRINT Q (F, C)
NEXT C
NEXT F
```

Las sentencias MAT READ y MAT PRINT tratan con valores "por filas"; es decir la primera fila de la matriz es tomada en cuenta, a continuación la segunda fila, la tercera luego, etc., tantas como filas haya. En consecuencia, cuando uno coloca valores en la sentencia DATA y él conoce que ellos serán procesados por una sentencia MAT READ, deberá poner aquellos valores en la sentencia DATA en conjuntos que contengan una fila de valores por conjunto.

Si las sentencias MAT READ y MAT PRINT no dicen nada sobre las dimensiones, se aplican las dimensiones dadas en la sentencia DIM.

Así en el ejemplo previo, las sentencias MAT READ y MAT PRINT se usan automáticamente en 5 x 5 dimensiones.

Pero un programa puede redefinir la matriz siendo leída e impresa de esta forma:

```
100 DATA 3, 6, 7, 9, 15, 17, 18, 21, 24, 30
110 DATA 33, 40, 42, 43, 45, 47, 49, 51, 58, 63
120 DATA 65, 78, 84, 85, 87
130 DIM Q (5, 5)
140 MAT READ Q (3, 4)
150 MAT PRINT Q
160 END
```

A pesar del hecho de que Q se dimensionó como una tabla 5 x 5, la sentencia MAT READ lo redimensiona, y Q será leído de 3 x 4 (3 filas por 4 columnas).

En la memoria, Q se carga como sigue:

Filas ↓ \ Columnas →	1	2	3	4
1	3	6	7	9
2	15	17	18	21
3	24	30	33	40

La instrucción MAT PRINT recuerda la redefinición e imprime la tabla como sigue:

3 15 24	6 17 30	7 18 33	9 21 40	

ya que no debe escribir una sentencia MAT PRINT de esta forma:

```
MAT PRINT Q (3, 4)
```

no funciona. Simplemente escribir

```
MAT PRINT Q
```

Si Q es redimensionada en la correspondiente sentencia MAT READ, la sentencia MAT PRINT usará automáticamente aquella definición. En otro caso, deberá usar la dimensión en la sentencia DIM.

Cuando se redefine una sentencia MAT READ, pueden usarse nombres en lugar de números en las redefiniciones –suponiendo que los valores que tenían antes han sido dados a aquellos nombres. Así

```
MAT READ (X, Y)
```

es lícito si los valores de X e Y han sido previamente asignados.

Ahora, ¿qué hay sobre las otras sentencias MAT?

CERO

Una matriz puede ser puesta toda ella a cero. He aquí un ejemplo:

```
10 DIM W (6, 6), X (10, 10)
20 MAT W = ZER
30 MAT X = ZER (4, 4)
40 MAT PRINT W, X
50 END
```

En este ejemplo, la tabla W entera es cargada con ceros; solamente 4 filas por 4 columnas de la tabla X son llenadas con ceros. El comando MAT PRINT usa totalmente la tabla 6 x 6 cuando imprime la tabla W y la tabla 4 x 4 redefinida cuando imprime la tabla X.

La sentencia "zero" puede escribirse como

```
MAT Y = ZER (M, N)
```

si a M y a N les han sido asignados valores previamente.

CONSTANTE UNO

Esta operación MAT trabaja de una forma semejante a la sentencia "zero" excepto, que la tabla es llenada con unos. La sentencia MAT se escribe como sigue:

```
MAT C = CON
```

ó

```
MAT D = CON (6, 7)
```

ó

```
MAT D = CON (M, N)
```

Una sentencia MAT PRINT asociada recuerda una redefinición de dimensión si se ha hecho una tal redefinición.

IDENTIDAD

Una matriz identidad puede establecerse escribiendo sentencias tales como

```
MAT H = IDN
```

ó

```
MAT J = IDN (5, 8)
```

ó

```
MAT P = IDN (P, Q)
```

Una matriz identidad es una tabla bidimensional teniendo unos a lo largo de la diagonal y ceros en todas las celdas restantes.

Cada redefinición de dimensión que podía haber sido hecha es recordada por la sentencia asociada MAT PRINT.

Unicamente en cuatro instrucciones MAT se pueden hacer las redefiniciones de la dimensión. Estas son READ, ZER, CON e IDN. Las restantes operaciones matriciales discutidas en este capítulo deben usar las dimensiones dadas en las sentencias DIM.

ADD – SUMA

La forma de esta sentencia matricial es

```
MAT E = A + B
```

Cada elemento de la tabla A es sumado al elemento correspondiente de la tabla B, y la suma se almacena en el elemento correspondiente de la tabla E. Los tres nombres –E, A y B– deben estar dimensionados. Estas dimensiones son las usadas actualmente.

SUBTRACT – RESTA

La forma de la sentencia es

```
MAT F = A – B
```

Cada elemento de la tabla B es restado del elemento correspondiente de la tabla A y la diferencia se almacena en el elemento correspondiente de la tabla F. Los tres nombres F, A y B deben estar dimensionados. Estas dimensiones son las usadas actualmente.

MULTIPLY – MULTIPLICACION

La forma de esta operación matricial es

```
MAT G = A * B
```

Cada elemento de la tabla A se multiplica por el elemento correspondiente de la tabla B, y el resultado se asigna al elemento correspondiente de la tabla G. Los nombres G, A y B deberán estar dimensionados. Estas son las dimensiones actuales usadas.

Unicamente pueden darse dos tablas en el lado de la derecha del signo igual, pero sin embargo, varias sentencias de la operación de multiplicación pueden darse en secuencia.

MULTIPLICACION POR UNA CONSTANTE

La forma de esta operación matricial es

```
MAT R = 3 * Y
```

o bien

```
MAT S = B * Y
```

o bien

```
MAT T = ((D + E)/F) * Y
```

Cada elemento de la tabla Y se multiplica por la constante dada en la sentencia (3 en el primer ejemplo, B en el segundo, (D + E)/F en el tercero), y el resultado se asigna al elemento correspondiente de la tabla nombrada en el miembro de la izquierda del signo (=). Todos los nombres de tablas R, S, T e Y deberán estar dimensionados. Estas dimensiones son las únicas usadas.

Las dos siguientes operaciones MAT son especialmente útiles a los matemáticos. Ellas son las sentencias matriciales INVERT y TRANSPOSE.

INVERT (INVERSA)

La forma de esta operación matricial es

```
MAT U = INV (A)
```

Se invierte la matriz A y el resultado se le asigna a la matriz U. Las matrices U y A deben ser dimensionadas. Estas dimensiones son las usadas.

Después de que una matriz ha sido invertida, el BASIC hace disponible un valor llamado DET. Este es el determinante de la matriz y puede imprimirse.

Ejemplo:

```
100 DATA 7, 4, 9, 2, 4, 5, 6, 4, 1, 8, 7, 6, 6, 1, 2, 8

110 DIM A (4, 4), B (4, 4)

120 MAT READ A

130 MAT B = INV (A)

140 PRINT DET

150 MAT PRINT B

160 END
```

TRANSPOSE (TRASPUESTA)

La forma de esta operación matricial es

```
MAT V = TRN (B)
```

La matriz B es traspuesta, y el resultado se asigna a la matriz V. Las matrices V y B deben ser dimensionadas. Estas dimensiones son las usadas.

MAT INPUT

La sentencia MAT INPUT permite la entrada de valores en un vector unidimensional, mientras el programa se está ejecutando actualmente. Al final de la input (entrada), una variable NUM contiene el número de valores que fueron entrados. Si el programador quiere no entrar valores, él sencillamente da el retorno de carro. NUM contiene entonces el valor cero. He aquí un ejemplo:

```
100 DIM D (20)

110 MAT INPUT D

120 IF NUM = 0 THEN 180

130 LET J = NUM
```

```
140 FOR K = 1 TO J

150 PRINT D (K)

160 NEXT K

170 GO TO 110

180 END
```

EJERCICIOS Y CUESTIONES

1. Definir el término matriz.
2. ¿A cuántas sentencias en BASIC sustituye la sentencia MAT READ?
3. Dar una sola sentencia MAT que pueda usarse en lugar de las cinco sentencias que comienzan en la línea 200:

```
10   DIM W (6, 6)
      .
      .
      .
      .
      .
      .
      .
      .
200  FOR K = 1 TO 6
210  FOR L = 1 TO 6
220  PRINT W (K, L)
230  NEXT L
240  NEXT K
250  END
```

4. ¿Cuándo una sentencia MAT READ o MAT PRINT necesita información sobre la dimensión?
5. Si una sentencia MAT READ o MAT PRINT no indica información de dimensión, ¿de dónde obtiene el programa las dimensiones que necesita?

6. Supongamos que Vd. necesita una matriz L cargada con estos valores:

L

1	5	9	13
2	6	10	14
3	7	11	15
4	8	12	16

Escribir la sentencia DATA correcta si ha de usarse una sentencia MAT READ L.

Supongamos que la sentencia DIM para L es

```
10 DIM L (4, 4)
```

7. Escribir la sentencia MAT que carga ceros en todas las celdas de una matriz F. La matriz F tiene esta sentencia DIM:

```
100 DIM F (10, 10)
```

8. ¿Pueden las dimensiones en una sentencia MAT ZER ser redefinidas de lo que ellas son en la sentencia correspondiente DIM?
9. ¿Qué sentencia MAT debe de usar si Vd. necesita almacenar unos en todas las células de una matriz P?
10. ¿Puede la información de la dimensión en una sentencia MAT CON ser redefinida de las dimensiones dadas en una sentencia correspondiente DIM?
11. ¿Qué sentencia MAT deberá usar si Vd. necesita establecer una matriz identidad?
12. ¿Qué cuatro tipos de sentencia MAT permite la redefinición de la información de dimensión?
13. Escribir una sentencia MAT que indique cómo las matrices R y S pueden sumarse.
14. Escribir una sentencia MAT que indique cómo la matriz V puede ser restada de la matriz U. Asignar el resultado a la matriz W.
15. Escribir una sentencia que haga que la matriz L sea multiplicada por la matriz M. Asignar el resultado a la matriz X.
16. ¿Qué hay de erróneo en el comando matricial?

```
500 MAT K = L * M * N
```

17. Dar dos sentencias matriciales que multipliquen las matrices P, Q y R y asignen el resultado a la matriz T.

18. Dar el comando matricial que multiplica la matriz W por 6.
19. Dar el comando matricial que invierte la matriz K y asigna el resultado a la matriz Y.
20. Dar el comando matricial que traspone la matriz C y asigna el resultado a la matriz D.
21. Suponemos que deseamos entrar (to input) desde el terminal en la matriz X los valores 9, 7, 9, 4 y 3. Suponemos que la sentencia DIM para X es

```
DIM X (20)
```

    Dar el comando matricial que permitirá que estos valores entren.
22. Cuando los valores mencionados en la cuestión 21, han entrado, ¿qué valor toma NUM?

CAPITULO **19**

Sentencias alfanuméricas

Hasta este momento, los valores de que nos hemos ocupado eran todos numéricos. Es posible trabajar con valores que sean *alfanuméricos*; es decir valores que no se usan para cálculos, sino para otros propósitos.

Consideremos este sencillo ejemplo:

```
10 LET A$ = "EN UN LUGAR DE LA"
20 LET B$ = "MANCHA"
30 LET C$ = "DE CUYO NOMBRE"
40 PRINT A$; B$; C$
50 END
```

El programa imprimirá estas palabras sobre el papel de la salida:

```
EN UN LUGAR DE LAMANCHADE CUYO NOMBRE
```

Como Vd. puede ver, en lugar de asignar valores numéricos a las celdas de la memoria, hemos asignado palabras. Las palabras asignadas estaban encerradas entre comillas. Los nombres de las celdas de la memoria usados incluyen el signo dolar ($). Ellas fueron A$, B$ y C$. Si Vd. pretende asignar una palabra, número de catálogo, dirección o algún otro mensaje alfanumérico a un nombre, dicho nombre deberá tener un signo de dolar $ añadido a él. En cualquier otro caso, las reglas que Vd. usa para crear nombres variables son todavía aplicables.

Estos nombres son correctos para usarlos con información alfanumérica:

```
P$
D3
L5$
Q$
```

Estos no lo son:

```
D       (necesita un signo de dolar)

DY$     (la D debe ser seguida por un dígito)

D3A$    (el nombre es demasiado largo)
```

Los valores que Vd. aisgna a un nombre pueden contener hasta 60 caracteres (más en algunos sistemas). Por ejemplo esta asignación es totalmente correcta:

```
100 LET W$ = "EN UN LUGAR DE LA MANCHA DE CUYO
              NOMBRE"
```

Cuando el usuario desea imprimir el anterior mensaje, puede dar el comando:

```
705 PRINT W$
```

Dando los espacios dentro de las marcas de comillas, se imprimen en las posiciones correspondientes de una línea de impresión. Consideremos, por ejemplo, cómo podemos mejorar el programa que fue presentado al comienzo de este capítulo. La forma en que se hicieron las asignaciones a A$, B$ y C2 da lugar a que algunas palabras funcionen juntamente. Para hacer la línea de salida más legible, necesitamos blancos entre las palabras LA y MANCHA, y entre MANCHA y DE. Podemos obtener estos blancos cambiando las sentencias de asignación en las líneas 10 y 20 para leerlo como sigue:

```
10 LET A$ = "EN UN LUGAR DE LA "

20 LET B$ = "MANCHA "
```

Observemos los únicos blancos siguientes a las palabras LA y MANCHA. Cuando le decimos al ordenador que funcione este programa, obtendremos esta salida:

```
EN UN LUGAR DE LA MANCHA DE CUYO NOMBRE
```

Existen otras formas de hacer el cambio. Por ejemplo, podíamos simplemente cambiar la línea 20 así:

```
20 LET GB$ = " MANCHA "
```

Ahora hay blancos antes y después de la palabra MANCHA.

Los valores alfanuméricos pueden ser leídos de la sentencia DATA. Estudiemos este ejemplo:

```
10 DATA LA INSTRUCCION, AL FINAL DEL
20 DATA PROGRAMA, DEBE CAMBIARSE
30 DATA PARA TERMINAR
40 READ A$, B$, C$, D$, E$
50 PRINT A$; B$; C$; D$; E$
60 END
```

El programa asigna LA INSTRUCCION a A$, AL FINAL DEL a B$, PROGRAMA a C$, DEBE CAMBIARSE a D$, y PARA TERMINAR a E$. Entonces se imprimen aquellos valores como sigue:

```
LA INSTRUCCIONAL FINAL DELPROGRAMADEBE CAMBIARSEPARA TERMINAR
```

Vd. puede ser que tenga algunos problemas de espaciado por la forma en que fueron leídos los valores de los datos. Alternativamente, podemos situar marcas entre comillas alrededor de cada uno de los valores de

los datos en la sentencia DATA. Aun cuando las comillas pueden no ser estrictamente necesarias. Cambiaremos algo la sentencia DATA:

```
10 DATA "LA INSTRUCCION", " AL FINAL DEL"

20 DATA " PROGRAMA", " DEBE CAMBIARSE"

30 DATA " PARA TERMINAR"
```

Los blancos se han insertado antes de la A en AL, de la P en PROGRAMA, de la D en DEBE y de la P en PARA. Estos blancos aparecerán sobre el papel de la salida cuando imprimamos A$ hasta E$. La salida se verá como ésta:

Los valores en una sentencia DATA están separados por comas. Si un valor de los datos es un item puramente numérico, o si tiene caracteres que pueden provocar confusión, el programador deberá colocar comillas alrededor de los valores. Por ejemplo, consideremos estos valores, que deseamos asignar a X$, Y$ y Z$:

```
10 DATA "3XLD", "TROY, N.Y.", "74 FOX AVE"
```

Las comas ordinariamente separan los valores de los datos, pero la coma en TROY, N.Y. no causa ninguna confusión puesto que el valor está entre comillas.

Una vez que ha sido leído un valor alfanumérico, uno puede consultarlo con una sentencia IF como la siguiente:

```
300 IF X$ = "JOHN WILLIAMS" THEN 600

310 IF Y$ = Z$ THEN 1600

320 IF Z$ > X$ THEN 2600
```

Un valor alfanumérico puede ser consultado para determinar cuándo es menor, igual, o mayor que algún otro valor. En BASIC un valor alfanumérico es mayor que otro valor alfanumérico si es mayor en la secuencia de caracteres. Esta escala designa que los dígitos son menores que todas las letras del alfabeto. Un dígito o una letra es mayor que otro si es naturalmente mayor en la escala. Así, 671 es mayor que 421 y JONES es mayor que BUTLER.

Vamos a intentar resolver un problema. Supongamos que tenemos estos valores en los datos en una sentencia DATA:

```
400 DATA ABE, JOHN, WILL, ANN, BETTY, TOM

410 DATA MARY, HELEN, SALLY, KEN, BEN, LUCY
```

Estos nombres identifican a los estudiantes en una clase de ciencias sociales. La clase está trabajando en cuatro proyectos. Tres estudiantes son asignados a cada proyecto. Así, los estudiantes ABE, JOHN y WILL lo son al primer proyecto; ANN, BETTY y TOM lo son al segundo, etc.

Nosotros necesitamos un programa que identifique los otros dos estudiantes de un proyecto cuando el nombre de un estudiante se ha entrado desde el teclado.

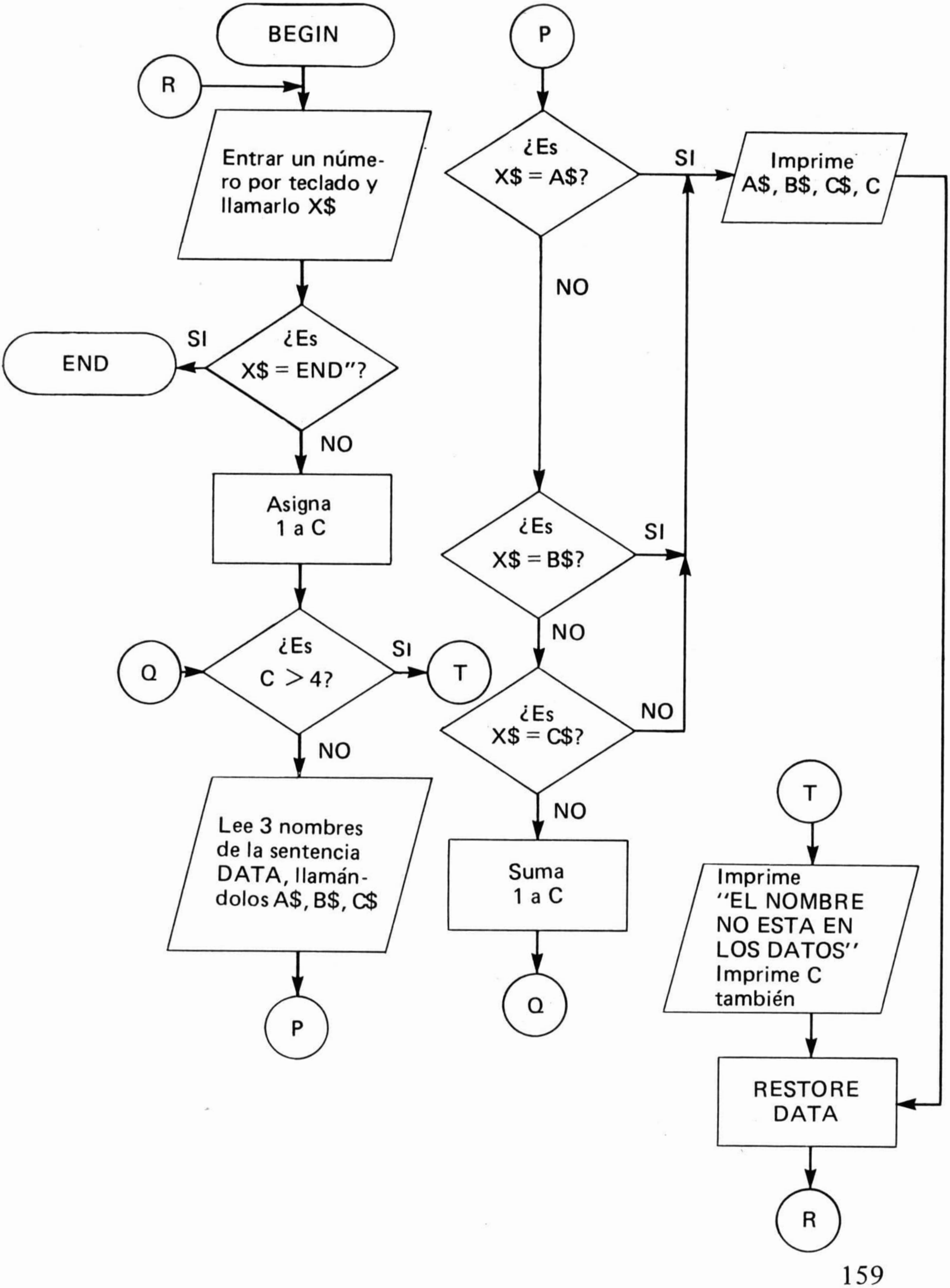
Ejemplo:

Si reteclea el nombre SALLY, el programa da los nombres de los estudiantes MARY, HELEN y SALLY. El programa deberá decir también en qué proyecto están implicados. El diagrama de flujo para este programa aparecerá a continuación.

He aquí el programa:

```
400 DATA ABE, JOHN, WILL, ANN, BETTY, TOM
410 DATA MARY, HELEN, SALLY, KEN, BEN, LUCY
420 INPUT X$
430 IF X$ = "FIN" THEN 550
440 FOR C = 1 TO 4
450 READ A$, B$, C$
460 IF X$ = A$ THEN 530
470 IF X$ = B$ THEN 530
480 IF X$ = C$ THEN 530
490 NEXT C
500 PRINT "EL NOMBRE NO ESTA EN LOS DATOS", X$
510 RESTORE$
520 GO TO 420
530 PRINT A$, B$, C$, C
540 GO TO 510
550 END
```

La sentencia RESTORE$ en la línea 510 restaura datos alfanuméricos en un programa precisamente como RESTORE restaura los datos


BEGIN
R
Entrar un número por teclado y llamarlo X$
¿Es X$ = END"?
SI
END
NO
Asigna 1 a C
Q
¿Es C > 4?
SI
T
NO
Lee 3 nombres de la sentencia DATA, llamándolos A$, B$, C$
P
P
¿Es X$ = A$?
SI
NO
Imprime A$, B$, C$, C
¿Es X$ = B$?
SI
NO
¿Es X$ = C$?
NO
NO
Suma 1 a C
Q
T
Imprime "EL NOMBRE NO ESTA EN LOS DATOS" Imprime C también
RESTORE DATA
R

numéricos. RESTORE Y RESTORE$ actúan independientemente uno de otro. Una sentencia DATA puede contener valore numéricos y alfanuméricos. Los tipos de los nombres a los que aquellos valores son asignados deben ser los adecuados.

Ejemplo:

```
10 DATA LOS, MEJORES, 3345, TRABAJOS, ESTAN, "3345",
   AQUI
```

puede ser leída con la sentencia READ siguiente:

```
20 READ E$, F$, X, G$, H$, K$, L$
```

El valor X puede usarse en los cálculos, pero K$ no puede serlo. (X es un valor numérico puro, mientras K$ es alfanumérico.)

Las tablas alfanuméricas pueden establecerse cuando se precisen.

Ejemplo:

```
10  DATA MARY, ALLEN, ROBERT, INGALLS, OLIVER, JAVIER
20  DATA FRED, ANDERSON, RICHARD, ILLY, NELSON, AMES
30  DATA LUCY, MAPES, NED, CONNERS, PETE, MILLS
40  DATA FRED, TIMS
50  DIM F$ (10), L$ (10)
60  FOR N = 1 TO 10
70  READ F$ (N), L$ (N)
80  NEXT N
90  INPUT X$
100 IF X$ = "HECHO" THEN 180
110 FOR K = 1 TO 10
120 IF X$ = F$ (K) THEN 160
130 NEXT K
140 PRINT "NOMBRE NO ENCONTRADO", X$
150 GO TO 90
160 PRINT X$, F$ (K), L$ (K)
170 GO TO 90
180 END
```

Este programa da el apellido de una persona cuando se entra el nombre a través del terminal de la máquina de escribir.

EJERCICIOS Y CUESTIONES

1. ¿Cómo debe construir un nombre que sea válido para guardar un valor alfanumérico en vez de numérico?
2. ¿Cuántos caracteres puede contener un valor alfanumérico?
3. ¿Pueden valores alfanuméricos ser enteramente numéricos?, es decir, ¿es esta sentencia correcta?

```
505 LET P$ = "4563"
```

4. ¿Los valores alfanuméricos pueden compararse?; es decir, ¿es esta sentencia correcta?

```
600 IF K$ > W$ THEN 800
```

5. ¿Dónde se halla el error en esta sentencia IF?

```
830 IF M$ TEST CASE THEN 1500
```

6. ¿Pueden usarse en el cálculo los valores alfanuméricos?
7. ¿Pueden los valore alfanuméricos darse en sentencias DATA?
8. Estudiar esta sentencia DATA:

```
1000 DATA JAN, 25, 1776
```

¿Cuántos valores hay en la sentencia?

9. Verdadero o falso. Para leer los valores de la sentencia DATA señalada en la cuestión 8 puede usarse esta sentencia READ:

```
1100 READ P6$, X, Y
```

10. Verdadero o falso. La sentencia DATA en la cuestión 8 puede ser correctamenreescrita:

```
1000 DATA "JAN", 25, 1776
```

11. Supongamos que necesitamos asignar una fecha al nombre alfanumérico L$. ¿Puede hacerse de esta forma?:

```
2000 LET L$ = "JAN 25, 1776"
```

12. Con referencia a la cuestión 11, ¿podemos asignar la misma fecha a L$ de esta forma?

```
3000 DATA "JAN 25, 1776"
3010 READ L$
```

13. ¿Qué hace la sentencia RESTORE$?
14. ¿Puede Vd. establecer tablas alfanuméricas si su programa los exige?
15. Señalar lo que este programa imprimirá:

```
10 LET A$ = "ESTO ES"
20 LET B$ = "UNA PRUEBA"
30 PRINT A$; B$
40 PRINT B$; A$
50 END
```

16. Escribir un programa que imprima los tres valores alfanuméricos ROGERS, WILLIAMS, BENSON en orden alfabético.
17. Escribir un programa que imprima los 10 valores alfanuméricos KINDER, SCHULE, MEIN, DER, HIER, KNABE, ALLES, ANFANG, IST, SCHWER en orden alfabético.
18. Escribir un programa que mantenga una conversación con un estudiante. El programa pregunta al estudiante nombre, edad y su asignatura favorita. El entonces imprime los tres items de información obtenidos desde el principio con un comentario sobre ellos.

CAPITULO **20**

Las sentencias GOSUB, RETURN y ON

La sentencia GOSUB, es muy semejante a la sentencia simple GO TO. La única diferencia es que en la primera, cuando el programa es instruido para ir al nuevo punto en el programa, el programa almacena el punto desde donde fue hecho el salto, regresando después automáticamente a dicho punto. Las sentencias GO TO, por otra parte, obligan a un programa a continuar procesando en el punto a donde se hizo el salto.

He aquí un esquema sucinto de un programa señalando cómo trabajan los GOSUB:

```
10 ~~~~~~~~
20 ~~~~~~~~
30 ~~~~~~~~
40 GOSUB 2000
50 ~~~~~~~~
60 ~~~~~~~~
70 ~~~~~~~~
80 GOSUB 2000
90  ~~~~~~~~
100 ~~~~~~~~
110 ~~~~~~~~
120 ~~~~~~~~
130 GOSUB 2000
140 ~~~~~~~~
150 ~~~~~~~~
160 GO TO 3000
2000 ~~~~~~~
2010 ~~~~~~~
2020 ~~~~~~~
2030 ~~~~~~~
2040 RETURN
3000 END
```

Las líneas onduladas indican sentencias que no están relacionadas a esta discusión y por consiguiente no se señalan con detalle.

El programa comienza en la línea 10 y ejecuta las sentencias en BASIC hasta que se alcanza la línea 40. El programa salta entonces a la línea 2000 (GOSUB 2000). El programa ejecuta las sentencias que comienzan con 2000 y halla la sentencia RETURN en la línea 2040. Esta sentencia hace que el programa vuelva atrás a la línea 50. Observemos que la línea 50 es la línea que sigue inmediatamente a la línea 40, línea que contiene la sentencia GOSUB.

Comenzando en la línea 50 el programa avanza ahora a la línea 80 donde encuentra otra sentencia GOSUB, GOSUB 2000. El programa todavía salta a la línea 2000 y progresa desde este punto a la línea 2040. La sentencia RETURN allí encontrada hace que el programa salte a la línea 90, que es la sentencia que sigue inmediatamente a la segunda sentencia GOSUB.

Desde 90, el programa avanza ahora a la línea 130 donde se encuentra un tercer GOSUB. El programa salta todavía a la línea 2000 y cuando la sentencia RETURN se encuentra por tercera vez, el programa vuelve a la línea 140.

Finalmente, el programa ejecuta todas las sentencias entre 140 y 160. La sentencia GO TO en la línea 160 indica al ordenador que salte directamente a la sentencia END de la línea 3000. Es importante que una sentencia GO TO aparezca antes de la sentencia 2000; en otro caso el programa debería una vez más ir al grupo de sentencias que comienzan en la línea 2000. En este momento, la sentencia RETURN en la línea 2040 debería causar dificultad porque la sentencia RETURN no puede ser equiparada a ninguna GOSUB dada previamente.

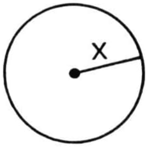
La sentencia GOSUB puede emplearse para indicar al ordenador que vaya a un grupo de sentencias conocido como una Subrutina. Las subrutinas se usan para proporcionar un grupo de instrucciones que puede ser accesible desde diversas partes del programa. Estudiemos un ejemplo: Supongamos que tenemos esta sentencia DATA;

```
100 DATA 18, 1, 14, 2, 7, 3, 4, 17, 2, 8, 2
110 DATA 10, 3, 7, 32, 1, 14, 3, 8, 9999, 0
```

Nuestro programa tiene que leer dos valores de datos usando la sentencia READ siguiente:

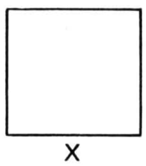
```
120 READ X, C
```

X es un valor numérico, mientras C es un código que dice lo que debe ser hecho con el valor. Si el valor de C es 1, X actúa como el radio de una circunferencia, y el programa debe calcular el área del círculo:



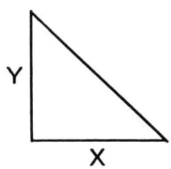
A = 3.1416 * X ↑ 2

Si el valor de C es 2, X actúa como la longitud de un lado de un cuadrado y el programa debe calcular el área del cuadrado:



A = X * X

Si el valor de C es 3, X actúa como la longitud de un lado de un triángulo rectángulo. El programa debe obtener el valor siguiente en la sentencia DATA que calcula el área del triángulo:

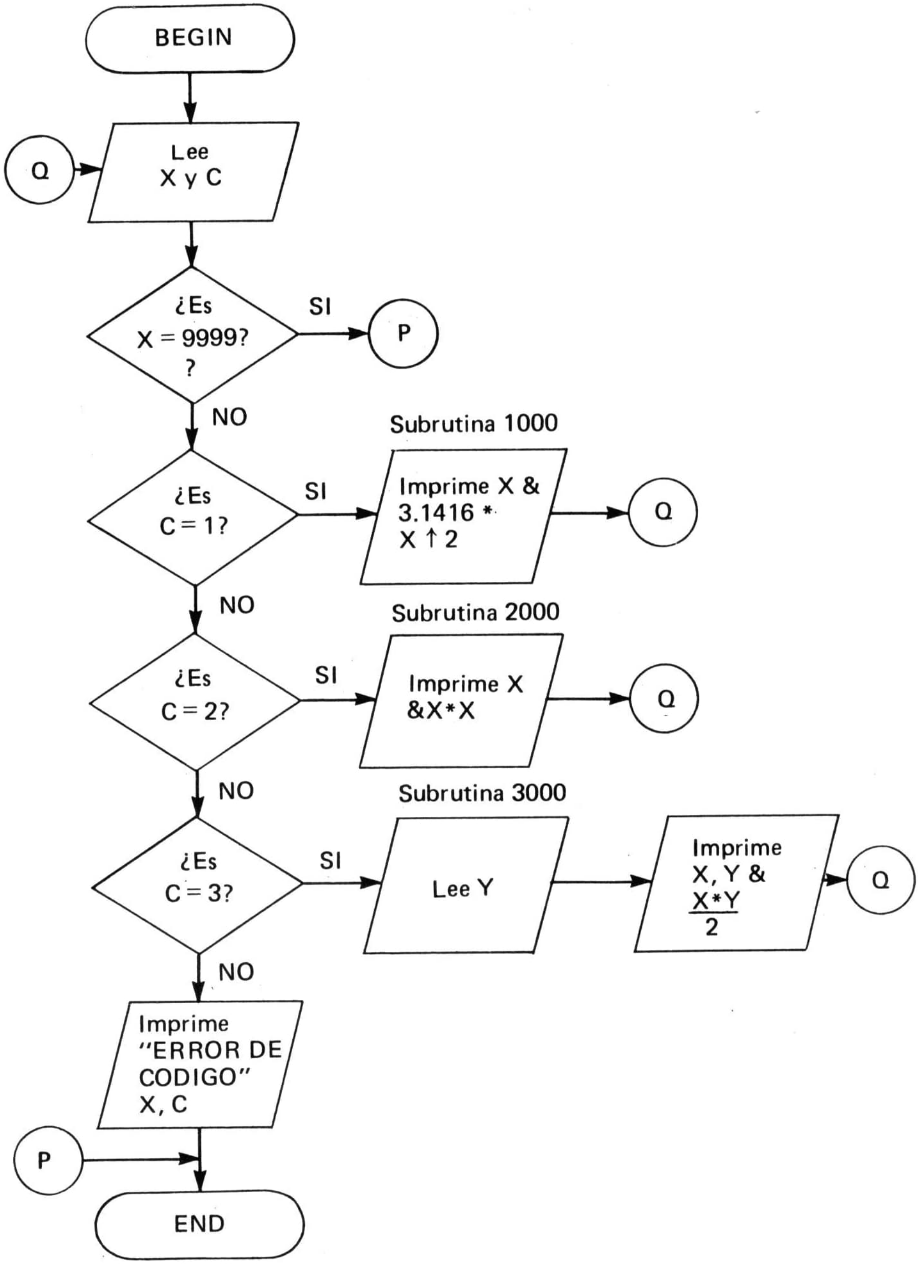


$$A = \frac{X * Y}{2}$$

Podemos utilizar el diagrama de flujo de la página siguiente.

El programa correspondiente es:

```
100 DATA 18, 1, 14, 2, 7, 3, 4, 17, 2, 8, 2
110 DATA 10, 3, 7, 32, 1, 14, 3, 8, 9999, 0
120 READ X, C
130 IF X = 9999 THEN 4000
140 IF C = 1 THEN 200
150 IF C = 2 THEN 300
160 IF C = 3 THEN 400
170 PRINT "ERROR DE COGIGO", X, C
180 GO TO 4000
```


BEGIN
Q
Lee
X y C
¿Es
X = 9999?
?
SI
P
NO
Subrutina 1000
¿Es
C = 1?
SI
Imprime X &
3.1416 *
X ↑ 2
Q
NO
Subrutina 2000
¿Es
C = 2?
SI
Imprime X
&X*X
Q
NO
Subrutina 3000
¿Es
C = 3?
SI
Lee Y
Imprime
X, Y &
X*Y
2
Q
NO
Imprime
"ERROR DE
CODIGO"
X, C
P
END

```
200  GOSUB 1000
210  GO TO 120
300  GOSUB 2000
310  GO TO 120
400  GOSUB 3000
410  GO TO 120
1000 PRINT X, 3.1416 * X ↑ 2
1010 RETURN
2000 PRINT X, X * X
2010 RETURN
3000 READ Y
3010 PRINT X, Y, (X * Y)/2
3020 RETURN
4000 END
```

El programa daría esta salida:

18	1017.88			
14	196			
7	4	14		
17	289			
8	64			
10	7	35		
32	3217			
14	8	56		

Podríamos haber escrito este programa con mayor sencillez sin usar subrutinas. Aun cuando la solución de este problema no exige actualmente subrutinas, lo escribimos de esta forma simplemente para demostrar cómo funcionan las subrutinas.

Las subrutinas pueden llamar a otras subrutinas. La sentencia RETURN dirige siempre al programa hacia detrás, a la sentencia inmediatamente siguiente al GOSUB que llamó a la subrutina. Puede haber cualquier número de subrutinas en un programa accesible por GOSUBs.

Otra forma de escribir el programa ejemplo dado superiormente es usando la sentencia ON. He aquí el programa:

```
100  DATA 18, 1, 14, 2, 7, 3, 4, 17, 2, 8, 2
110  DATA 10, 3, 7, 32, 1, 14, 3, 8, 9999, 0
120  READ X, C
130  ON C GO TO 1000, 2000, 3000
140  PRINT "ERROR DE CODIGO", X, C
150  GO TO 4000
1000 PRINT X, 3.1416 * X ↑ 2
1010 GO TO 120
2000 PRINT X, X * X
2010 GO TO 120
3000 READ Y
3010 PRINT X, Y, (X * Y)/2
3020 GO TO 120
4000 END
```

La sentencia ON espera que la variable que sigue a la palabra ON tenga un valor entero 1, 2, 3, etc. Si el valor asignado al nombre es 1, el programa saltará a la primera línea cuyo número se indica después de las palabras GO TO; si el valor es 2, a la segunda línea cuyo número se indica, etc.

Puede haber varios números de línea siguiendo las palabras GO TO pero el valor del código, deberá ser capaz de alcanzar todas ellas.

Por ejemplo si tenemos

```
50 ON K GO TO 100, 150, 200, 250, 300
```

El valor de K deberá en un instante u otro ser igual al valor entero 1, 2, 3, 4 ó 5. El valor del código no será nunca negativo ni tampoco un

valor mayor que el número de números de línea que siguen a GO TO. Si se presentan estas condiciones incorrectas, el programa no salta a un área alejada del programa; en su lugar va a la siguiente sentencia en secuencia.

Si el código que sigue a ON es un número decimal (por ejemplo, 3.6) el programa trunca la parte fraccionaria del código y usa el valor que pueda. Así, en nuestro ejemplo, cuando el código es 3.6 se quita la parte fraccionaria del código y el 3 restante actúa como el código final.

EJERCICIOS Y CUESTIONES

1. ¿Qué es una subrutina?
2. ¿Cómo se diferencia el comando GOSUB del comando GO TO?
3. ¿Qué hace la sentencia RETURN?
4. ¿Por qué los programas deben saltar alrededor de las subrutinas?
5. ¿Puede un programa saltar a la misma subrutina desde distintos puntos del programa?
6. ¿Puede una subrutina llamar a otra subrutina?
7. ¿Qué hace la sentencia ON?
8. Estudiar esta sentencia ON:

```
300 ON X GO TO 1200, 1300, 1400, 1500
```

   ¿Qué posibles valores tendrá X?
9. En la cuestión 8 ¿qué acontece si X no tiene los valores 1, 2, 3 ó 4?
10. ¿Puede tener Vd. más de una subrutina en un programa accesible por el comando GOSUB?
11. Verdadero o falso. Estas sentencias ON independientes entre sí son todas correctas:

```
3000 ON F GO TO 120, 130, 160, 190, 310

4000 ON P GO TO 4035, 3050

5000 ON E3 GO TO 6000, 7000, 8000

6000 ON G GO TO 30, 40, 60, 65, 75, 85, 90, 95
```

12. Consideremos esta sentencia ON:

```
350 ON L GO TO 80, 100, 150
```

Si el valor de L es 2.3, ¿dónde irá el programa?

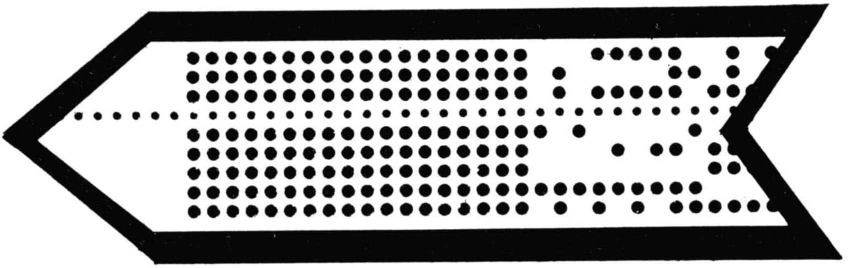
13. Usando la sentencia ON en la cuestión 11, encontrar dónde irá el programa si el valor de L es 2.8.
14. Usando la sentencia ON en la cuestión 11, hallar dónde irá el programa si el valor de L es 4.

CAPITULO **21**

Trabajando en cinta de papel

Muchos terminales proporcionan la facilidad de perforar programas sobre una cinta de papel. La clase de cinta que uno puede obtener, y el procedimiento actual para obtenerlo varía de un terminal a otro. Este capítulo se refiere solamente a la cinta de papel del Teletipo Modelo 33.

Este es el tipo de cinta de papel que nosotros estamos discutiendo:



La cinta de papel perforada puede usarse para:

1. Guardar programas.
2. Entrar programas en el sistema en lugar de escribir manualmente la información línea por línea.

Supongamos que se desea almacenar sobre la cinta de papel un programa que está creado y salvado en el sistema. El procedimiento es éste:

1. Poner ON (encender) la unidad de perforación de la cinta de papel.
2. Simultáneamente pulsamos las cuatro teclas CTRL, SHIFT, RPT y P. Así, situar hacia abajo las llaves hasta que Vd. obtenga 20 ó 30 caracteres. Con este procedimiento se dejan algunos blancos al principio de la cinta que Vd. puede usar para escribir manualmente el nombre del programa, fecha, etc.

3. Pulsar simultáneamente las dos teclas RUBOUT y RPT y dejarlas así hasta que Vd. ha obtenido alrededor de 20 caracteres RUBOUT sobre la cinta de papel.
4. Poner OFF a la unidad de perforación.
5. Pulsar la tecla RETURN.
6. Escribir LIST y a continuación dar el retorno de carro. Una fracción de un segundo después poner la unidad de perforación de la cinta de papel ON.
7. Esperar hasta que el sistema ha perforado completamente su programa sobre la cinta de papel.
8. A continuación dar alrededor de 20 caracteres más RUBOUT.
9. Entonces poner la unidad de perforación OFF. Dar el retorno de carro. Rasgar la cinta de papel. Dar una etiqueta a la cinta y guardarla para una aplicación futura.

Supongamos que Vd. debe crear una cinta de papel fuera de línea. Seguir estos pasos:

1. Poner el switch del teletipo en LOCAL.
2. Pulsar las cuatro llaves CTRL, SHIFT, RPT y P como se hizo antes y mantenerlos así hasta que se obtengan alrededor de 20 ó 30 caracteres. Si se tiene un área en blanco al comienzo de una cinta para etiquetarla a mano.
3. Pulsar las teclas RPT y RUBOUT y mantenerlas así hasta que alrededor de 20 caracteres RUBOUT han sido perforados.
4. Dar el retorno de carro.
5. Escribir una línea de su programa.
6. Dar una línea de alimentación usando la tecla LINE FEED.
7. Dar el retorno de carro.
8. Dar un carácter XOFF (pulsando las llaves CTRL y S simultáneamente).
9. Dar un carácter RUBOUT.
10. Repetir los pasos 5 hasta 9 con la frecuencia que sea necesaria para constuir su programa.
11. Al final de la cinta, poner unos 20 RUBOUTs adicionales.
12. Cortar la cinta, etiquetarla y poner el teletipo OFF (apagarlo).

Los caracteres de control de borrado (←) pueden darse en la forma usual. Los controles de borrado de líneas, sin embargo, deberán usarse como sigue:

1. Escribir la línea (para ser borrada).
2. Dar control de borrado de línea (pulsando las teclas CTRL y X).
3. Dar un carácter XOFF (pulsando CTRL y S).
4. Dar un carácter RUBOUT.
5. Escribir la línea corregida.
6. Dar una alimentación de línea pulsando la tecla LINE FEED.
7. Dar un retorno de carro.
8. Dar otro XOFF carácter.
9. Dar un carácter RUBOUT.

Para leer una cinta de papel en el sistema de tiempo compartido, seguir los pasos siguientes:

1. Situar la cinta de papel sobre la lectora de cinta de papel de forma que la cabeza lectora permanezca directamente sobre los caracteres RUBOUT.
2. Situar el switch de leer sobre la lectora de la cinta de papel en la posición FREE.
3. Escribir NEW. El sistema responderá READY.
4. Escribir la palabra TAPE.
5. Cuando el sistema escribe READY, poner el conmutador de la lectora a START. La cinta entera será leída.
6. Dar un retorno de carro.

Pueden presentarse algunas dificultades. En estos casos el usuario tomará varios tipos de acciones dependiendo de cuál es el problema. Si el sistema parece "desvanecerse", después de que ha leído una cinta, escribir un carácter WOFF (pulsando CTRL y S). Si el sistema no responde, probar a llamar su atención dando un retorno de carro o un BREAK.

Tratar con la cinta de papel es un poco delicado; pero después de que Vd. ha creado unas pocas cintas y las haya leído en el sistema, Vd. probablemente no tendrá ninguna dificultad ulterior.

Una vez que la cinta de papel ha sido dominada, su uso le ahorrará dinero. Para entrar al tiempo compartido, una cinta perforada fuera de línea es más barato leerla porque el sistema lee la cinta de papel más deprisa que cuando reteclea. Para salida, los programas guardados en la cinta no cuestan nada, mientras que los programas almacenados en el sistema cuestan al menos $2.00 por mes y por programa. Si Vd. hace

funcionar un programa con escasa frecuencia, guardándolo sobre la cinta de papel es definitivamente más barato; lo lee solamente cuando se necesite.

EJERCICIOS Y PROBLEMAS

1. ¿Por qué una cinta de papel es útil como medio de entrada mientras opera en el modo de tiempo compartido?
2. ¿Por qué se puede desear guardar un programa sobre una cinta de papel mejor que en el sistema de tiempo compartido?
3. Prepare un programa fuera de línea conteniendo al menos diez sentencias; a continuación leerlo en el sistema de tiempo compartido.
4. Copiar un programa del espacio de trabajo en la cinta de papel; a continuación leerlo en el sistema de tiempo compartido. Comprobar los resultados para estar seguro de que el proceso se hizo bien.

CAPITULO **22**

Funciones definidas

En capítulos anteriores, hemos discutido varias funciones construidas en lenguaje BASIC. Estas son SIN, COS, SQR, etc. Aquí, por ser conveniente, presentamos una lista completa de las funciones *incorporadas* junto con un breve resumen de cómo pueden usarse. (La información que aparece dentro de los paréntesis para cada una de estas funciones se llama *argumento de la función.*)

Función	*Finalidad*
SIN (X)	Calcula el seno de X. X puede ser un número constante o un nombre o expresión en BASIC. X debe venir expresado en radianes.
COS (X)	Calcula el coseno de X. X debe ser un valor como se describe en SIN (X).
TAN (X)	Calcula la tangente de X. X debe ser un valor como se describe en SIN (X).
ATN (X)	Calcula el arco tangente de X. X debe ser un número constante o un nombre en BASIC o una expresión. El arco tangente de X viene dado en radianes.
EXP (X)	Calcula e^X. El valor de X debe ser un número constante o un nombre en BASIC o una expresión. El valor de e es 2.718281828...
ABS (X)	Obtiene el valor absoluto de X. X debe ser un valor como se describe en EXP (X).
LOG (X)	Calcula el logaritmo natural de X. X debe ser un valor como se describe en EXP (X).
SQR (X)	Calcula la raíz cuadrada de X. X debe ser un valor como se describe en EXP (X).
RND (X)	Obtiene un número aleatorio. El número queda comprendi-

Función	*Finalidad*
	do entre 0 y 1. Puede ser cero, pero nunca puede efectivamente ser uno. Escriba RANDOM en la primera línea de su programa si Vd. desea números no repetibles.
INT (X)	Obtiene el mayor entero (parte entera) de X. X debe ser un valor como se describe en EXP (X).

Si una función que a Vd. le gustaría usar en un programa no está en la lista de funciones dada anteriormente, Vd. puede definir su propia función. Cerca del comienzo de su programa puede Vd. escribir una sentencia semejante a ésta:

```
100 DEF FNA (L) = (SIN (L) + COS (L))/4
```

Vd. ha inventado una función llamada FNA. Esta función puede ahora ser usada en cualquier sitio de su programa en que Vd. necesite sumar el seno y el coseno de un valor y dividir el resultado por 4.

El motivo de que esta función no exista ya en BASIC es que los programadores tienen poca necesidad de tal función. Sin embargo, Vd. puede necesitar esta función muchas veces para un programa especial con el que está trabajando. Si lo desea, el BASIC le permite a Vd. inventar una función para usarla durante la duración de un único programa. Si Vd. necesitase la misma función en otro programa, tendría que definirla otra vez de nuevo totalmente en este nuevo programa.

Habiendo definido una función, Vd. se halla ahora interesado en cómo usarla. Una función definida se usa de la misma forma que Vd. usaba una función incorporada; es decir, si Vd. tiene funciones definidas como FNB, FND y FNQ, puede usarlas de las siguientes formas:

```
200 LET F = FNB (2.5) + SIN (D)

210 LET H = FND (L + M)

220 LET K = FNQ (N) – SQR (P)
```

Los valores situados entre los paréntesis de las funciones definidas pueden ser valores numéricos constantes o nombres en BASIC o expresiones. Estos valores constituyen los argumentos de las funciones definidas.

Como puede ver, las funciones deberán definirse cerca del comienzo del programa –o al menos antes de que la función se use actualmente en el programa–. Se da nombre a una función, el nombre que se le da

deberá comenzar con FN. El carácter tercero y final en el nombre puede ser cualquier letra del alfabeto.

En la definición de la función, la única letra situada entre paréntesis es un carácter ficticio o virtual. Esto no tiene otra finalidad que indicar cómo los valores actuales, cuando se dan, deberán ser manejados. Así la definición para FNA dada anteriormente era

```
100 DEF FNA (L) = (SIN (L) + COS (L))/4
```

L no es un valor actual. Es un definidor, señalando que cuando otro valor tal como (P – Q)/R se da para usar en la función tal como:

```
350 LET G = FNA ((P – Q)/R)
```

se calcula primero el valor de (P – Q)/R; a continuación el seno y coseno del valor anterior. Aquellos valores se suman y el resultado se divide por 4. En resumen, L representa a (P – Q)/R en este ejemplo.

Después en el programa, si la sentencia

```
850 LET Z = FNA (S)
```

fuese ejecutada, el valor de S debería ser manejado en la misma forma en que L ha sido manejado en la definición de una función; es decir el valor Z debería calcularse como

```
(SIN (S) + COS (S))/4
```

Como con SQR, SIN, COS y todas las restantes funciones incorporadas, solamente un valor, puede usarse en ambas, la definición de una función y su uso.

EJERCICIOS Y CUESTIONES

1. ¿Qué es una función incorporada?
2. Listar las funciones incorporadas discutidas en este capítulo.
3. ¿Qué hace la función incorporada SIN?
4. ¿En qué medida deberán estar los valores que Vd. da a la función SIN?
5. ¿Qué entiende por argumento de una función?
6. ¿Cuáles son las tres formas que puede tomar el argumento de una función?
7. ¿Qué hace la función incorporada COS?

8. ¿Qué hace la función incorporada TAN?
9. ¿Qué hace la función incorporada ATN?
10. ¿Cómo es el valor proporcionado por la función ATN?
11. ¿cuáles son los primeros dígitos del valor matemático e?
12. ¿Qué hace la función incorporada EXP?
13. Verdadero o falso. Las dos sentencias

```
100 LET P = EXP (W)
100 LET P = 2.71828 ↑ W
```

dan resultados casi idénticos.
14. ¿Qué hace la función incorporada ABS?
15. Verdadero o falso. La sentencia

```
200 LET X = ABS (Y)
```

siempre asigna un valor positivo a X.
16. ¿Qué hace la función incorporada LOG?
17. ¿Qué hace la función incorporada SQR?
18. Verdadero o falso. Las dos sentencias

```
200 LET M = SQR (F)

300 LET M = F ↑ (.5)
```

dan resultados casi idénticos.
19. Verdadero o falso. La sentencia

```
400 LET N = G ↑ (.333333)
```

da la raíz cúbica de G y la asigna a N.
20. Verdadero o falso. Las dos sentencias

```
500 LET L = SQR (SQR (D))

500 LET L = D ↑ (.25)
```

dan casi idénticos resultados.
21. ¿Qué hace la función incorporada RND?
22. ¿Por qué convendría poner el comando RANDOM como primera sentencia de su programa?

23. Verdadero o falso. El valor cero puede ser dado actualmente por la función RND pero el valor uno nunca se da.
24. ¿Qué hace la función incorporada INT?
25. ¿Cuál es el valor asignado a K cuando se ejecuta la sentencia siguiente?

```
250 LET K = INT (3.2)
```

26. ¿Cuál es el valor asignado a K si el valor indicado en la cuestión 25 es 3.8?
27. ¿Qué hace el comando DEF?
28. ¿Puede Vd. usar una función definida exactamente en la misma forma en que Vd. usa una función incorporada tal como SQR o LOG?
29. Verdadero o falso. Todas las siguientes cuatro funciones contienen argumentos correctos.

```
350 LET D = SQR (246.3)

360 LET E = LOG (L)

370 LET M = COS ((A + B)/C)

380 LET Y = FNG (SQR (X))
```

30. Verdadero o falso. Vd. puede definir más de una función en un programa en BASIC.

CAPITULO **23**

Sentencias de manejo de ficheros

Existen seis tipos de sentencias que permiten el manejo de ficheros. Son éstas: FILES, READ, WRITE, SCRATCH, RESTORE e IF.

Pero antes de nada tenemos que entender lo que es un fichero. Un fichero es una serie de líneas almacenadas que contienen números. Estos números son conservados bajo algún nombre de la misma forma que se conservan los programa.

Suponemos que deseamos guardar este fichero:

Número de empleado	*Sueldo horario*
265	3.15
300	4.05
315	2.95
335	3.80
450	5.50
480	4.30
515	3.75
575	4.15

Nosotros podemos escribirlo de igual forma que cuando escribíamos un programa:

```
100 265, 3.15,
200 300, 4.05,
300 315, 2.95,
400 335, 3.80,
500 450, 5.50,
600 480, 4.30,
700 515, 3.75,
800 575, 4.15,
```

Los números 100, 200, etc. son números de línea, –los ficheros requieren números de línea exactamente como los necesitan los programas–. Sobre cada línea del fichero, el primer número qué sigue al número de línea representa el número del empleado y el número siguiente representa el sueldo horario del trabajo. Observemos que las comas siguen a todos los valores de los datos, pero no siguen a los números de línea. Ahora podemos conservar el fichero escribiendo SAVE. Cualquier nombre que elijamos como nombre de un fichero es totalmente correcto. Este nombre puede tener hasta seis caracteres formados por letras del alfabeto y dígitos. Supongamos que a este fichero lo hemos hombrado MASTER.

Ahora podemos escribir otro fichero conteniendo esta información:

Número de empleado	*Horas trabajadas*
265	38.5
300	41.5
315	40.0
335	39.0
450	40.5
480	40.0
515	42.5
557	37.5

El fichero puede ser escrito y guardado de la misma forma que MASTER fue salvado. Podemos escribir:

```
100 265, 38.5,
200 300, 41.5,
300 315, 40.0,
400 335, 39.0,
500 450, 40.5,
600 480, 40.0,
700 515, 42.5,
800 575, 37.5,
```

Supongamos que hemos nombrado a este fichero TRANS.

Ahora podemos escribir un programa que lea una línea del fichero MASTER y una línea del fichero TRANS y calcule la paga bruta. Entonces registraremos la información en un nuevo fichero llamado NEWFIL.

Nosotros damos el programa y a continuación lo explicamos:

```
100 FILES MASTER; TRANS; NEWFIL
110 SCRATCH #3
120 IF END #1 THEN 210
130 READ #1, E1, R
140 READ #2, E2, H
150 IF E1 <> E2 THEN 200
160 LET G = H * R
170 WRITE #3, E1, H, R, G
180 PRINT E1, H, R, G
190 GO TO 120
200 PRINT "ERROR EN FICHERO. NUMS EMPLEADOS NO
    COINCIDEN."
210 END
```

La sentencia FILES en la línea 100 da los nombres de los ficheros que se usarán en las líneas que siguen. Ella deberá ser la primera sentencia del programa. Observemos que los nombres de los ficheros están separados por punto y coma.

En el programa cuando #1 es referenciado (véase línea 130), el fichero referido es el primero en la sentencia FILES, es decir MASTER; cuando #2 es referenciado (véase línea 140) el fichero referenciado es el segundo en la sentencia FILES, es decir TRANS; y cuando #3 es referenciado (véase línea 170), el fichero referido es NEWFIL.

Es importante entender que los ficheros existen siempre en uno de los dos modos *–read* y *write–*. Un fichero en modo READ puede ser leído, pero no grabado; un fichero en modo write puede ser grabado pero no leído. Por consiguiente, con el fin de que un fichero pueda ser grabado, deberá ser cambiado del modo read al modo write. La sentencia SCRATCH en la línea 110 realiza este trabajo para el fichero #3 (NEWFIL). Si un fichero no se cambia del modo read al modo write, el programa no lo grabará. Un mensaje de error será dado cuando se haga tal intento.

El comando SCRATCH puede darse en cualquier parte de un programa con tal que se dé antes de que el programa intente grabar el fichero.

Si un fichero está en modo write y se desea cambiar el fichero al modo read, el programador da el comando RESTORE:

```
RESTORE #3
```

Tener presente que inicialmente todos los ficheros están en modo read y el comando RESTORE no es necesario antes de que sean leídos los ficheros correspondientes.

La sentencia IF END consulta a un fichero para determinar si hay más información que puede ser leída. La forma de la sentencia es

```
IF END #1 THEN 800
```

Si la sentencia detecta que el final del fichero ha sido alcanzado (es decir, que no hay más datos en el fichero para ser leídos), el programa salta a la línea 800; en otro caso, el programa va a la siguiente sentencia en secuencia.

En el programa ejemplo, la sentencia IF END está en la línea 120. La sentencia comprueba el fichero #1 MASTER, para determinar cuándo deberá ejecutarse la sentencia aplicada READ en la línea 130. El programa va a la siguiente sentencia en secuencia si el final del fichero #1 no ha sido encontrado; él va a la sentencia 210 si ha sido encontrado el final del fichero #1.

La comprobación de "fin de fichero" está situada antes que la sentencia aplicable READ. Se dará un mensaje de error si ha sido hecho un intento de hacer leer un fichero que no tiene datos en él para ser leídos.
La sentencia WRITE en la línea 170 provoca la entrada de una línea de salida en el fichero referenciado NEWFIL. La línea salida tendrá un número de línea (el sistema de la línea 10 como el primer número de línea y los números de línea aumentan de diez en diez unidades) seguidos por los valores nombrados. En el ejemplo aquellos valores son E1, H, R y G.

El programa imprime también la misma línea que ha sido registrada en el fichero. La sentencia PRINT es opcional. Un programa puede escribir en un fichero, pero no imprimir la misma línea, o él puede imprimir simplemente una línea sin grabarla en el fichero, o él puede hacer ambas cosas.

Ahora ocupémonos del programa mismo. El programa lee una línea de MASTER y una línea de TRANS. El tiene entonces la siguiente información en su memoria:

```
De MASTER    E1 = 265
             R = 3.15

De TRANS     E2 = 265
             H = 38.5
```

El valor 265 representa el número del empleado encontrado en la primera línea de cada fichero. Este programa requiere que los dos números coincidan. Si ello no sucediese el programa supone que existe una dis-

crepancia seria y el programa se detiene. Si los números coinciden, entonces 3.15 encontrado en el fichero master (maestro) representa la cuantía del sueldo por hora del empleado 265, y 38.5 representa las horas trabajadas la última semana por el mismo empleado. El programa calcula la paga bruta, G, del empleado 265. Entonces él graba cuatro valores en NEWFIL; número del empleado, horas trabajadas, cuantía del sueldo y paga bruta. El programa imprime también una línea sobre el papel de la salida dando la misma información.

El programa vuelve a la sentencia IF END para repetir el ciclo si existen más datos en el fichero #1. Cuando los datos se han agotado en el fichero #1, el programa salta a la sentencia END y el trabajo ha terminado.

Para determinar cuándo el programa ha creado bien NEWFIL, se puede llamar a NEWFIL de esta forma:

```
OLD NEWFIL
```

y entonces hacer que el sistema lo liste. El programador escribe

```
LIST
```

NEWFIL contiene la información actualizada concerniente a los empleados cuyos registros fueron encontrados en los ficheros MASTER y TRANS. NEWFIL puede usarse por consiguiente como la entrada de datos para algún otro programa que el usuario puede escribir.

Los comandos que manejan los ficheros difieren apenas de un sistema a otro. Si los comandos indicados en este capítulo no trabajan sobre su sistema, tomar contacto con su representante, su instructor o un compañero usuario. Encontrará que las diferencias son escasas y fácil aprenderlas.

EJERCICIOS Y CUESTIONES

1. ¿Qué se entiende por el término fichero?
2. ¿Cómo debe construir un fichero para usarlo en un programa en BASIC?
3. ¿Cuáles son los dos modos de operar con un fichero?
4. ¿Cuál es el modo oeprativo inicial de un fichero?
5. ¿Qué comando debe dar cuando desea cambiar el modo oeprativo de un fichero de lectura a grabación?
6. ¿Cómo actúa la sentencia FILES?

7. ¿Qué sentencia usará cuando quiera obtener información de un fichero? Dar un ejemplo.
8. ¿Qué sentencia usará cuando quiera registrar información en un fichero? Dar un ejemplo.
9. ¿Cómo puede consultar a un fichero para determinar cuándo hay información adicional en él que puede leerse?
10. Cuando está grabando en un fichero, ¿puede Vd. imprimir también la misma información sobre el papel de salida?
11. ¿Cómo actúa el comando RESTORE?
12. ¿Cómo puede examinar la salida de un fichero para verificar que estaba registrada en él la información correcta?

CAPITULO **24**

Una palabra final

Vd. ha completado ahora un estudio del lenguaje de programación en BASIC. Este lenguaje es ideal para alguien que comienza porque es fácil de aprender y sencillo de usar. En el futuro Vd. emprenderá posiblemente, el estudio de otros lenguajes como FORTRAN, COBOL, PASCAL o PL/I, Vd. verá que el BASIC ofrece una flexibilidad cercana a la de los otros lenguajes. Un hecho sorprendente es que el BASIC permite a un programador el llevar a cabo ciertos trabajos que son o bien imposibles o de mayor dificultad en otros lenguajes.

Nosotros le deseamos buena suerte con su programación en BASIC.

Respuestas

CAPITULOS 1-23. EJERCICIOS Y CUESTIONES

Capítulo 1

1. BASIC.
2. El terminal puede estar situado tan cercano como a 10 pies y tan lejano como a 5.000 millas.
3. Un ID es el nombre de identificación del usuario.
4. ALGOL, FORTRAN, PL/I.
5. El usuario escucha un tono alto de timbre.
6. El ordenador da la fecha y la hora de conexión.
7. El ordenador quiere conocer si el usuario desea recuperar un viejo conjunto de instrucciones (OLD) o si el usuario desea dar al ordenador un nuevo conjunto de instrucciones (NEW).
8. El usuario da el nombre del programa que quiere crear.
9. Cada instrucción en BASIC se llama una *sentencia* en BASIC.
10. Un número de línea es un entero que precede a cada instrucción en BASIC.
11. No. Ellos pueden aumentar en pasos cualesquiera regulares o irregulares que el usuario quiera.
12. DATA, LET, READ, IF, PRINT, GO TO, END, RUN y BYE. La combinación GO TO se considera una palabra.
13. RUN y BYE son comandos del sistema.
14. No. Porque los comandos del sistema no forman parte del lenguaje BASIC.
15. El usuario debe pulsar la tecla CARRIAGE RETURN. Esta acción hace que el ordenador acepte la última sentencia escrita.
16. *Ejecutar* un programa significa hacerlo funcionar y obtener así los resultados.
17. BYE.

Capítulo 2

1. El BASIC se emplea en el modo conversacional de tiempo compartido.
2. El modo es *conversacional* porque el usuario y el ordenador son interactivos; es decir, el usuario escribe algo y el ordenador responde. El término tiempo-compartido indica que el tiempo del ordenador es compartido por otros muchos usuarios.
3. Falso. El ordenador informa al usuario de dicho error cuando el usuario escribe RUN. El error puede entonces a su vez corregirse.
4. Véase la respuesta a la cuestión 2.
5. El ordenador es muy rápido y un usuario no le pregunta mucho durante el tiempo en que está conectado; en consecuencia el ordenador tiene una labor pequeña al obedecer los comandos de sus numerosos usuarios de tiempo compartido.
6. Las tasas mensuales están basadas, en la cantidad de tiempo en que un usuario está actualmente conectado al ordenador, en el tiempo actual usado del ordenador, y en el coste del alquiler de un terminal.
7. Aproximadamente \$ 5.00 por hora.
8. Aproximadamente \$ 0.25 por segundo.
9. Aproximadamente \$ 150.00 al mes.
10. El usuario puede comenzar a construir un nuevo programa, o puede comenzar a usar un viejo programa que el sistema ha recuperado para él.
11. El menor número de línea que puede usarse es 1; el mayor es 99999.
12. El programa es automáticamente puesto en secuencia con arreglo a los números de línea.
13. El símbolo (←) se emplea para que el carro salte un espacio hacia atrás.
14. ←←← * Z/Q.
15. Deberá pulsar las teclas CTRL y X simultáneamente para borrar la entrada entera.
16. RUN.
17. RUN, BYE, LIST, OLD, NEW y SAVE.
18. El ordenador comprobará el programa por si aparecen errores. Si no hay ninguno, ejecutará el programa.
19. El ordenador acepta la segunda sentencia.
20. El usuario escribe el número de línea de la sentencia y entonces da retorno de carro.
21. Para insertar una sentencia previamente olvidada, escoge un número de línea no usado entre los números de línea existentes; él entonces entra la sentencia.

Capítulo 3

1. Un tipo de sentencia se identifica por la primera palabra en la sentencia.
2. LET, IF, PRINT, GO TO, END.
3. Verdadero.
4. La sentencia LET se usa para asignar un número dado o el resultado de un cálculo a un nombre.
5. El término assignar significa la donación de un valor a una celda de la memoria del ordenador.
6. Los nombres en BASIC pueden constar de una sola letra del alfabeto o de una letra sola seguida de un dígito.
7. Verdadero. Todos los nombres son correctos.
8. Falso. Todos los nombres son incorrectos.
9. 3W—En BASIC los nombres constarán de un carácter alfabético único, seguido por un dígito único.

   FL—Dos caracteres alfabéticos son siempre incorrectos.

   A34—Unicamente se permiten dos caracteres en un nombre en BASIC.
10. A la celda de memoria T se le da el valor 6.2.
11. 4.6 es sumado a la celda de memoria T. (En BASIC, es posible que el mismo nombre aparezca en los dos lados del signo igual.)
12. 4, 9.3, 3.6, 7.6. (El valor de L es 4; el de P, 9.3; el de Q, 3.6; y el de R, 7.6.)
13. 300—LET está escrita incorrectamente.

    320—El nombre DTL en BASIC es incorrecto.

    330—El valor dado a N tiene dos puntos decimales.

    350—DONE es un tipo de sentencia incorrecta en BASIC.
14. Verdadero. Los valores de A, B y C serán todos cero puesto que ningún valor les fue asignado a dichos nombres.

Capítulo 4

1. La sentencia IF se usa con esta finalidad.
2. = igual.
> mayor que.
< menor que.
>= mayor o igual que.
<= menor o igual que.
<> no igual.
3. El programa irá a la sentencia identificada por el número que sigue a THEN.
4. El programa irá a la siguiente sentencia en secuencia.

5. Las tres formas son nombre de la celda de memoria, valor numérico actual y expresión aritmética.
6. Los símbolos aritméticos son + (suma), – (resta), * (multiplicación), / (división) y ↑ (elevar a una potencia).
7. Las palagras GO TO pueden sustituir a THEN en algunos sistemas.
8. Falso. THEN GO TO 600 no está permitido. Puede escribirse THEN 600 ó GO TO 600.
9. Verdadero. El programa irá siempre a la sentencia 620.
10. El programa imprimirá sobre cuatro líneas separadas: 4 (el valor de Z), 4 (el valor de Z), 2 (el valor de X) y 3 (el valor de Y).
11. El programa imprimirá sobre dos líneas separadas: 5 (el valor de X) y 4 (el valor de Y).
12. El programa imprimirá: 15 (el valor de Y).

Capítulo 5

1. La sentencia PRINT se usa con este fin.
2. Verdadera.
3. Cinco zonas disponibles: Zona 1 (1-15), Zona 2 (16-30), Zona 3 (31-45), Zona 4 (46-60) y Zona 5 (61-72).
4. Ver la respuesta a la cuestión 3 anterior.
5. Para esta finalidad se utilizan la coma y el punto y coma.
6. 205– Los valores de L, F y P se imprimirán en las zonas 1, 2 y 3.

   206– Los valores de L, F y P se imprimirán con un máximo de dos espacios entre cada valor.

   207– El valor de L se imprimirá en la zona 1.
   El valor de F deberá comenzar a imprimirse en la zona 2.
   El valor de P se imprimirá a un máximo de dos espacios siguientes al valor asignado a F.
7. El valor de P comenzará en la posición de impresión 31.
8. El valor de P comenzará en la posición de impresión 11.
9. El valor de P comenzará en la posición de impresión 21.
10. Se deberán usar comillas (“ ”).
11. El ordenador imprime una línea en blanco.
12. Las comillas siguientes a la palabra ES faltan.
13. Los dos puntos (:) es un símbolo ilegal usado para separar nombres de celdas de la memoria.
14. 10 PRINT F * L.

15. Verdadero.
16. La sentencia puede escribirse:

```
400 PRINT "     TECLEAR VALORES DE R Y W"
```

Hay cinco espacios entre las comillas primeras y la letra T.
17. El ordenador imprimirá 43 en una línea y 24 y 2 sobre la línea siguiente.
18. El valor L comenzará en la posición de impresión 17.
19. La sentencia correcta es

```
1000 PRINT A; B; C; D; E; F; G;
```

Capítulo 6

1. Una tranferencia condicional es un salto del ordenador a un punto distante en el programa dependiendo de que una cierta condición sea verdadera o falsa.
2. 10 IF A > B THEN 100.
3. El término tranferencia incondicional significa que un programa salta incondicionalmente a una sentencia distante.
4. 20 GO TO 90.
5. La auténtica última sentencia de cualquier programa en BASIC es la sentencia END. Ejemplo:

```
5000 END
```

6. Verdadera.
7. Verdadera.
8. Verdadera.
9. La sentencia END no es la última sentencia del programa.
10.
```
100 LET X = 1
110 IF X > 5 THEN 150
120 PRINT X
130 LET X = X + 1
140 GO TO 110
150 PRINT "FIN DE LA EJECUCION"
160 END
```

Capítulo 7

1. Las sentencias DATA y READ.
2. Proporcionar una lista de valores para ser usados en el programa; estos valores son accesibles por las sentencias READ.

3. Tomar un valor no usado de la sentencia DATA y asignar dicho valor al nombre que sigue a READ.
4. Sí. Ejemplo:

```
500 READ P, Q, R
```

5. La sentencia DATA puede estar en cualquier posición antes de la sentencia END.
6. No existe límite al número de sentencias DATA que pueda tener un programa.
7. Sí.
8. Un bucle es una serie de sentencias en BASIC que se ejecutan repetidas veces.
9. OUT OF DATA in nnnnn. Las letras nnnnn representan el número de línea donde la sentencia READ asociada está situada.
10. Verdadero. El programa funcionará agotando los datos antes de alcanzar END; sin embargo, imprimirá todos los datos.
11. Verdadero. Sin embargo tampoco alcanzará nunca END.
12. Verdadero. Sin embargo, solamente un valor será leído e impreso.
13. El programa imprimirá OUT OF DATA in nnnnn. (Véase también la respuesta a la cuestión 9).
14. Señalar el fin de los datos.
15. El valor ficticio no deberá ser accidental o inadvertidamente igual a uno de los valores actuales que deben procesarse.
16.
```
10  READ A
20  IF A = 1000 THEN 999
30  PRINT A
40  GO TO 10
100 DATA 7, 14, –3, 8, 4, 11, 1000
999 END
```
17.
```
10   READ A
20   IF A = 1000 THEN 9999
30   PRINT A, A ↑ 3
40   GO TO 10
100  DATA 7, 14, –3, 8, 4, 11, 1000
9999 END.
```
18. Notación exponencial significa que los valores numéricos se expresan en potencias de 10. Ejemplo:

```
200 LET B = 3.5E3
```

El valor asignado a B es 3.5×10^3, ó 3500.

19. La letra E significa "por 10 elevado a la potencia". En la cuestión 18 el valor asignado a B es 3.5 por 10 elevado a la potencia 3.

20. 12.56 es 1.256E1; −43.67 es −4.367E1; 8.0 es 8.0E0. Existen otras muchas formas de escribir estos números con notación exponencial.
21. Verdad. El valor es 92630000.
22. Verdad. El valor numérico es .00636.
23. Verdad. Los valores numéricos son 8, 760000., 4.76, −.036 y 999.
24. La sentencia REM permite al programador insertar una observación en su programa. Esto le permitirá dar un mensaje a él mismo o algún otro observador diciéndoles lo que está aconteciendo en una particular parte del programa.
25. TAB. Ejemplo:

```
150 PRINT TAB (5); "*"
```

26. En la posición de impresión 40.
27. En la posición de impresión 50.
28. En la posición de impresión 40.
29. La sentencia PRINT USING puede usarse para posicionar los valores en el papel de salida. Usándola, pueden ser alineados los puntos decimales sobre líneas consecutivas.
30. Sí. Los programas pueden saltar a la sentencia END.
31. No. La sentencia END deberá siempre ser la última sentencia de cualquier programa en BASIC.

Capítulo 8

1. Las sentencias READ, DATA, LET, PRINT, IF, GO TO y END.
2. Las sentencias IF y GO TO.
3. Las sentencias DATA y READ.
4. La sentencia LET.
5. La sentencia PRINT.
6. Al final del programa.
7.

```
10   READ A, B
20   IF A > B GO TO 500
30   LET A = B
40   READ B
50   IF B = 999 GO TO 700
60   GO TO 20
500  PRINT "NUMEROS SIN SECUENCIA"
550  GO TO 9999
700  PRINT "NUMEROS EN SECUENCIA CRECIENTE"
1000 DATA 9, 15, 16, 18, 21, 25, 31, 36, 999
9999 END
```

8.
```
300 DATA 9, 2, 6, 5, 6, 3, 4, 1, 8, 0, 0, 0
310 READ A, B, C
320 IF A = 0 THEN 420
330 IF A < B THEN 360
340 LET S = B
350 GO TO 370
360 LET S = A
370 IF S < C THEN 400
380 PRINT C; "  ES EL MENOR"
390 GO TO 310
400 PRINT S; "  ES EL MENOR"
410 GO TO 310
420 END
```

9.
```
350 DATA 3, 21, 4, 9, 6, 18, 4, 1, 14, 19, 17, 1000
360 READ L
370 READ X
380 IF X = 1000 THEN 420
390 IF X <= L THEN 370
400 LET L = X
410 GO TO 370
420 PRINT L; "  ES EL MAYOR"
430 END
```

10.
```
350 DATA 3, 21, 4, 9, 6, 18, 4, 1, 14, 19, 17, 1000
360 READ L
370 LET S = L
380 READ X
390 IF X = 1000 THEN 470
400 IF X > L THEN 450
410 IF X < S THEN 430
420 GO TO 380
430 LET S = X
440 GO TO 380
450 LET L = X
460 GO TO 380
470 PRINT S; "  ES EL MENOR"
480 PRINT L; "  ES EL MAYOR"
490 END
```

11.
```
800 DATA 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 9000
810 LET Z = 0
820 LET W = 0
830 READ X
840 IF X = 9000 THEN 900
```

```
850 IF X = 0 THEN 880
860 LET W = W + 1
870 GO TO 830
880 LET Z = Z + 1
890 GO TO 830
900 PRINT Z; "  NUMERO DE CEROS"
910 PRINT W; "  NUMERO DE UNOS"
920 END
```

12.
```
200 DATA 8, 4, 3, 4, 0, 8, 0, 3, 4, 8, 4, 3, 2000
210 LET K1 = 0
220 LET K2 = 0
230 LET K3 = 0
240 LET K4 = 0
250 READ A
260 LET K1 = K1 + 1
270 READ X
280 IF X = A THEN 430
290 LET B = X
300 LET K2 =  K2 + 1
310 READ X
320 IF X = A THEN 450
330 IF X = B THEN 470
340 LET C = X
350 LET K3 = K3 + 1
360 READ X
370 IF X = A THEN 490
380 IF X = B THEN 510
390 IF X = C THEN 530
400 LET D = X
410 LET K4 = K4 + 1
420 GO TO 550
430 LET K1 = K1 + 1
440 GO TO 270
450 LET K1 = K1 + 1
460 GO TO 310
470 LET K2 = K2 + 1
480 GO TO 310
490 LET K1 = K1 + 1
500 GO TO 360
510 LET K2 = K2 + 1
520 GO TO 360
530 LET K3 = K3 + 1
540 GO TO 360
```

```
550 READ X
560 IF X = 2000 THEN 680
570 IF X = A THEN 620
580 IF X = B THEN 640
590 IF X = C THEN 660
600 LET K4 = K4 + 1
610 GO TO 550
620 LET K1 = K1 + 1
630 GO TO 550
640 LET K2 = K2 + 1
650 GO TO 550
660 LET K3 = K3 + 1
670 GO TO 550
680 PRINT K1, A
690 PRINT K2, B
700 PRINT K3, C
710 PRINT K4, D
720 END
```

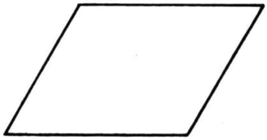
13.
```
10  LET A = 0
20  LET B = 1
30  LET C = A + B
40  IF C = 1597 THEN 80
50  LET A = B
60  LET B = C
70  GO TO 30
80  LET K = 1
90  PRINT C
100 LET K = K + 1
110 IF K > 10 THEN 160
120 LET A = B
130 LET B = C
140 LET C = A + B
150 GO TO 90
160 END
```

14.
```
10 LET S = 0
20 LET X = 0
30 IF X > 100 THEN 70
40 LET S = S + X
50 LET X = X + 2
60 GO TO 30
70 PRINT S
80 END
```

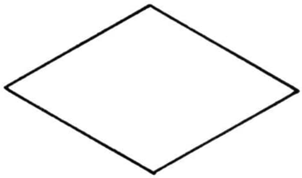
Capítulo 9

1. (1) El programador planea las instrucciones que él dará finalmente al ordenador. (2) Proporciona documentación para uso posterior.

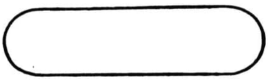
2.



3.

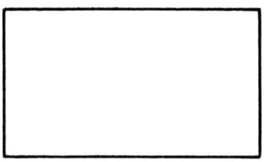


4.

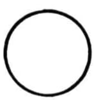


5. El símbolo BEGIN se situará en la esquina superior izquierda.

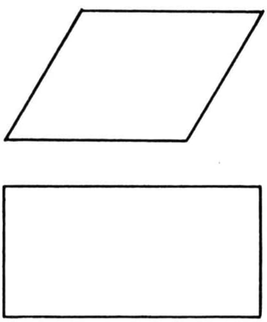
6.



7.

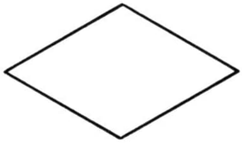


8.



```
10 READ W
30 PRINT W
```

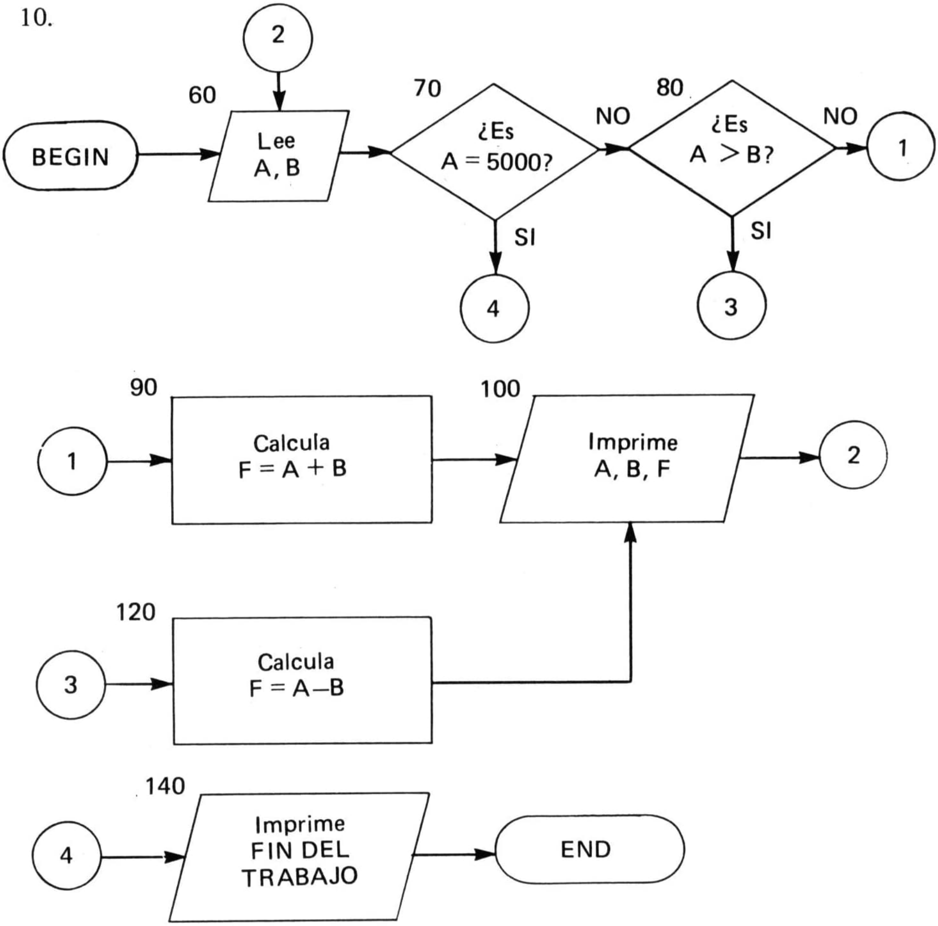
No hay sentencia de asignación en el programa.



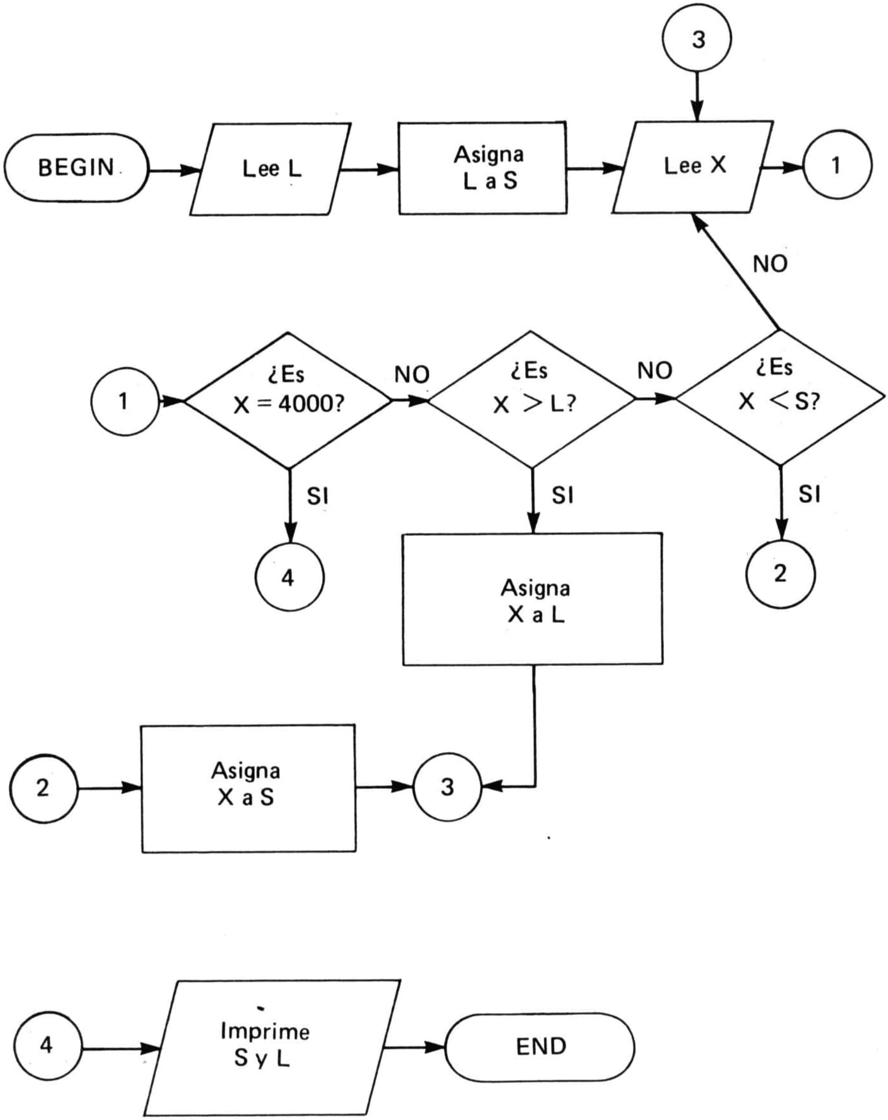
20 IF W = 6 THEN 500

9.
```
10    LET K = 5
20    LET F = K ↑ 3
30    IF K = 10000 THEN 500
40    PRINT W, F
50    LET K = K + 3
60    GO TO 20
500   PRINT "FIN DEL TRABAJO"
9999  END
```

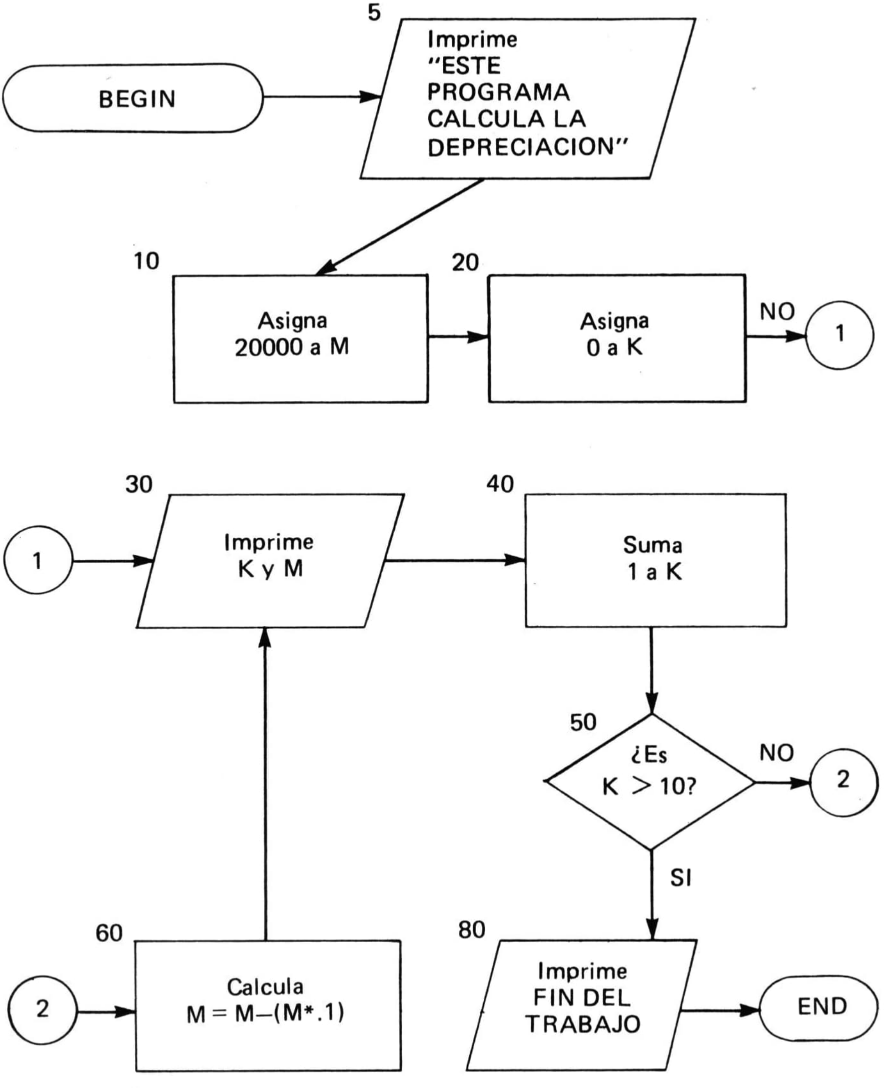
10.



11.


3
BEGIN
Lee L
Asigna
L a S
Lee X
1
NO
1
¿Es
X = 4000?
NO
¿Es
X > L?
NO
¿Es
X < S?
SI
SI
SI
4
Asigna
X a L
2
2
Asigna
X a S
3
4
Imprime
S y L
END

12.



Capítulo 10

1. + Suma
   – Resta
   * Multiplica

/ Divide
↑ Eleva a una potencia.

2. Los paréntesis se usarán para clarificar los cálculos que deberán hacerse.
3. Expresiones dentro de paréntesis tienen el mayor nivel de prioridad.
4. Las exponenciaciones tienen el segundo nivel de prioridad.
5. Multiplicaciones y divisiones, en cualquier orden que aparezcan, tienen el tercer nivel de prioridad.
6. Adiciones y sustracciones, en cualquier orden que aparezcan, tienen el cuarto y menor nivel de prioridad.
7. SIN—para obtener senos trigonométricos.

   COS—para obtener cosenos trigonométricos.

   SQR—para obtener raíces cuadradas.

   ABS—para obtener valores absolutos.

   LOG—para obtener logaritmos naturales.
8. El ordenador multiplicará G * K primero (G * K aparece dentro de un conjunto de paréntesis).
9. El ordenador elevará al cubo el valor de la celda de memoria L.
10. Cuando hay duda, usar paréntesis.
11. 10 LET G = (W – S)/P + SQR (M + N).
12. La L está en el lado erróneo del signo igual.
13. El valor asignado a Z es –482.

$$Z = 4 - 3^5 \times \frac{4}{2}$$

$$Z = 4 - 245 \times 2$$

$$Z = 4 - 486$$

$$Z = -482$$

14. 10 LET R1 = (–B + SQR (B ↑ 2 – 4 * A * C))/(2 *A)
    20 LET R2 = (–B – SQR (B ↑ 2 – 4 * A * C))/(2 * A)
15. 40 LET R1 = (–B + D)/L
    50 LET R2 = (–B – D)/L

Capítulo 11

1. Un comando del sistema es una instrucción que dice al ordenador lo que ha de hacer con un programa (RUN, SAVE, etc.). Un comando del sistema nunca está precedido por un número de línea.

2. No. Los comandos del sistema nunca están precedidos por números de línea.
3. El comando RUN dice al ordenador que ejecute el programa que el programador ha entrado. El ordenador hace funcionar al programa si no hay errores tipográficos en él.
4. Si un programa funciona, él puede dar todavía respuestas incorrectas. Por ejemplo el programador puede haber dado una o más ecuaciones incorrectas.
5. El comando LIST da un listado hasta el momento del programa que un usuario ha entrado. El listado refleja los cambios que pueden haber sido hechos.
6. El comando SAVE sitúa una copia del programa con que el usuario estaba trabajando en una memoria permanente en el sistema. Cuando el programa ha sido salvado, tiene un nombre de identificación único.
7. *Espacio temporal de trabajo* es un área en la memoria del ordenador donde un programa es construido y posiblemente modificado. *Espacio permanente de trabajo* es un área donde una copia del programa en el espacio de trabajo es copiado y almacenado.
8. El comando del sistema NEW borra enteramente el espacio temporal de trabajo.
9. El sistema trae del espacio de trabajo permanente una copia del programa requerido. El programa se carga en el espacio temporal de trabajo.
10. No. El programa está todavía en el espacio de trabajo.
11. El espacio de trabajo temporal es borrado completamente y el sistema queda en condiciones de aceptar el nuevo programa a ser entrado.
12. El comando UNSAVE borra o limpia del espacio de trabajo permanente al programa que es nombrado.
13. Un mensaje de error puede darse si el programa, NUMANL, existía ya en el espacio de trabajo permanente.
14. Un mensaje de error puede darse si el programa, BUDGET, ya existía en el espacio permanente de trabajo.
15. Una persona debería usar REPLACE más bien que SAVE si él quisiera borrar un viejo programa ya salvado en el espacio de trabajo permanente bajo algún nombre y sustituirlo con el programa que él tiene en el espacio del trabajo temporal.
16. Sí. La persona debe esperar. El puede inadvertidamente borrar algún programa con el que ha estado trabajando y desea guardar.

Capítulo 12

1. Un bug es un error de programa; puede ser de escritura, tal como él escribe RAED cuando se pretendía que fuese READ, o puede ser lógico, tal como escribir GO TO 40 cuando se pretendía GO TO 30. Depuración es el proceso de encontrar errores y eliminarlos.
2. Verdadero. Ver la respuesta a la cuestión 1 anterior.

3. UNRECOGNIZABLE COMMAND IN LINE 500.
4. PUNCTUATION ERROR IN LINE 50.
5. Uno puede corregir un error en una sentencia reescribiéndola.
6. Uno puede borrar escribiendo el número de sentencia e inmediatamente dar el retorno de carro.
7. El programa escribe el mensaje OUT OF DATA IN y da el número de línea de la sentencia READ que causa el mensaje. Ejemplo:

   OUT OF DATA IN 40

   La línea 40 podría ser:

   40 READ A, B, C

8. A (one-shot job) una tarea de una vez es un programa que funcionará solamente una vez. Después de que las respuestas han sido obtenidas el programador no tiene ulterior interés en el programa.

   A (production job) una tarea de producción es un programa que funcionará periódicamente durante un tiempo relativamente largo después de haber sido depurado.
9. En una tarea de una vez, varias de las respuestas dadas por el programa son comprobadas manualmente con un calculador de pupitre. Las respuestas de los problemas escogidos que resultan del cálculo manual deberán comprobarse exactamente con las respuestas del ordenador.
10. En un trabajo de producción, las respuestas seleccionadas del ordenador son comprobadas otra vez a mano y los cálculos realizados junto a los mismos problemas. También, el nuevo programa es comprobado en paralelo al lado de un procedimiento manual que el programa puede sustituir.
11. Sentencias extra PRINT indican qué resultados temporales están siendo asignados a las variables en partes sensibles del programa.
12. Las sentencias extra PRINT usadas únicamente para el depurado, deben eliminarse cuando el programa funcione correctamente.

Capítulo 13

1. Un bucle es una parte de la codificación en un programa, que es ejecutada repetidas veces. El bucle puede constar de solamente unas pocas sentencias o de cientos de sentencias.
2. No todos los bucles son malos. Los bucles pueden ser deseables puesto que pueden precisarse idénticos procedimientos para procesar datos diferentes leídos o calculados.
3. El contador indica el número de veces que ha de ejecutarse el bucle.
4. Verdadero.

5. En un diagrama de flujo de la estructura standard de un bucle, el contador es consultado inmediatamente después de que le ha sido dado su valor inicial.
6. El tamaño del paso es el incremento que hay que sumar o restar del contador cada vez que el valor del contador cambia.
7. Si el tamaño del paso es positivo, el bucle se termina cuando el valor del contador excede el tamaño previamente definido para aquel contador. El contador no se usa con este valor excesivo.
8. Si el tamaño del paso es negativo, el bucle termina cuando el valor del contador se hace menor que el mínimo valor previamente definido.
9. Línea 200.
10. Línea 210.
11. Líneas 220 y 230.
12. Línea 240.
13. Un programa puede usar igualmente una sola sentencia en el cuerpo de un bucle, como muchos cientos, e incluso miles de sentencias.
14. Las sentencias FOR y NEXT automatizan la escribura de un bucle.
15. La sentencia FOR da el nombre del contador del bucle. Da también los valores inicial y final del contador y el tamaño de su paso.
16. La sentencia NEXT define la sentencia final del cuerpo del bucle. También causa un incremento en el valor del contador dado por el tamaño del paso.
17. Falso. No debe haber ninguna coma siguiendo a 300.
18. Línea 400–C

    Línea 450–C

    Línea 500–I Esta sentencia necesita un tamaño de paso negativo.

    Línea 550–C

    Línea 660–I Esta no es una sentencia en BASIC.

    Línea 650–I Esta sentencia necesita un tamaño de paso positivo.
19. Sí.
20. 55.
21. El contador es consultado inmediatamente después de cada aumento. Cuando su valor se hace menor que el valor mínimo dado en la sentencia FOR, la tarea (bucle) ha terminado.
22. El contador tomará y usará los valores 1, 3, 5, 7, ..., 99.
23.
```
10 FOR X = 10 TO 25
20 PRINT X, X ↑ 2, X ↑ 3, X ↑ 4
30 NEXT X
40 END
```
24.
```
10 LET S = 0
```

```
20 FOR K = 2 TO 24 STEP 2
30 LET S = S + K
40 PRINT S
50 NEXT K
60 END
```

Capítulo 14

1. La función INT da el mayor entero dentro de un valor dado. Así el entero de 21.6 es 21; el entero de –21.6 es –22.
2. La función RND da un número al azar. El número está comprendido entre 0 y 1. Puede ser efectivamente 0, pero nunca puede alcanzar completamente 1.
3. U = 4
   V = 4
   W = 4
   X = –10
   Y = –10
   Z = –10
4. `100 LET V = INT (Q + .5)`
5. `100 LET W = (T * 100 + .5)/100`
6. `300 IF INT (V/2) * 2 = V THEN 400`
7. `500 LET R = V – INT (V/B) * B`
8. Falso. No da 1.674.
9. Falso. No da 1.000.
10. Falso. El nombre X puede usarse en cualquier parte del programa.
11. La palabra RANDOM se da como la primera sentencia del programa BASIC.
12. `400 LET I = INT (RND (X) * 61) + 17`
13.
```
10 RANDOM
20 LET S = 0
30 FOR K = 1 TO 100
40 LET S = S + RND (X)
50 NEXT K
60 PRINT S/100
70 END
```
14.
```
10 RANDOM
20 LET S = 100
30 FOR K = 1 TO 1000
40 LET R = RND (X)
50 IF K = S THEN 70
60 GO TO 90
```

```
70  PRINT R
80  LET S = S + 100
90  NEXT K
100 END
```

15.
```
10 FOR K = 1 TO 10000
20 LET R = INT (RND (X) * 100) + 1
30 IF R = 53 THEN 50
40 GO TO 70
50 PRINT "EL NUMERO 53 SALIO EN EL ENSAYO"; K
60 GO TO 90
70 NEXT K
80 PRINT "EL NUMERO 53 NO SALIO EN LOS 10000 ENSAYOS"
90 END
```

16.
```
10 LET S = 0
20 FOR K = 1 TO 100
30 LET R = RND (X)
40 IF R > .5 THEN 60
50 LET S = S + 1
60 NEXT K
70 PRINT "NUMERO DE BOLAS NEGRAS"; S
80 PRINT "NUMERO DE BOLAS BLANCAS"; 100 - S
90 END
```

Capítulo 15

1. La sentencia RESTORE permite que los valores en una sentencia DATA sean leídos otra vez. RESTORE puede usarse con la frecuencia necesaria en conexión con la sentencia DATA.
2. Los valores pueden leerse solamente una vez.
3. El ordenador imprime OUT OF DATA IN y da el número de línea de la sentencia READ que causa el mensaje. Ejemplo:

   ```
   OUT OF DATA IN 40
   ```

4. El ordenador escribe una marca de interrogación.
5. El ordenador espera tres valores para L, F y V.
6. ESCRIBA UN NUMERO ENTERO, POR FAVOR.
7. El usuario escribirá en algún número entero.
8. El ordenador comprueba si el número escrito es 5.
9. Si el usuario escribe 5, el ordenador escribe

¡SI, ES EL NUMERO!

y a continuación se detiene.

10. Si el usuario no escribe 5, el ordenador escribe

NO ES ESTE.

y entonces señala al usuario que escriba otro número.

11. El ordenador escribe

TENGO UN NUMERO
¿PUEDE VD. ACERTARLO?
TECLEE SU INTENTO.

12. El ordenador espera que Vd. conjeture qué número tiene el ordenador.
13. Comprueba el número para ver si es igual a su número. También comprueba su número para ver cuándo es mayor o menor que su número.
14. El ordenador imprime

¡ENHORABUENA!

A continuación le pregunta si desea Vd. jugar otro juego.

15. El ordenador le dice a Vd. cuándo su conjetura es demasiado grande o demasiado. pequeña.
16. El ordenador escribe

¿QUIERE JUGAR OTRA VEZ?

Entonces le indica que Vd. escriba 1 si su respuesta es sí ó 0 si su respuesta es no.

17.
```
10  PRINT " ¡HOLA, YO SOY EL ORDENADOR!"
20  PRINT "¿DIGAME CUANTO ES 6 POR 8?
30  INPUT A
40  IF A=48 THEN 70
50  PRINT "NO, NO ES CORRECTO. INTENTELO DE NUEVO
    POR FAVOR"
60  GO TO 20
70  PRINT " ¡EXCELENTE!"
80  PRINT "¿DIGAME CUANTO ES 7 POR 9?"
90  INPUT A
100 IF A=63 THEN 130
110 PRINT "NO, NO ES CORRECTO, INTENTELO DE NUEVO
    POR FAVOR"
```

```
120 GO TO 80
130 PRINT " ¡ENHORABUENA!"
140 END
```

18.
```
10  READ X
20  IF X = 999 THEN 50
30  PRINT X + X
40  GO TO 10
50  RESTORE
60  READ X
70  IF X = 999 THEN 110
80  PRINT X * 3
90  GO TO 60
100 DATA 18, 92, 47, 36, 62, 999
110 END
```

Capítulo 16

1. La sentencia DIM reserva espacio para un vector. Establece el nombre y el tamaño del vector.
2. Un vector o tabla es una disposición de valores numéricos. La organización de la estructura puede ser con una dimensión (una lista) o en dos o más dimensiones.
3. La sentencia DIM da el nombre y el tamaño del vector.
4. Los valores iniciales son normalmente ceros.
5. Verdadero.
6. Falso. Algunos problemas son casi imposibles de resolver sin vectores.
7. Falta el nombre del tercer vector.
8. El nombre es L.
9. El tamaño es 200.
10. Un subíndice es un puntero. El indica qué miembro de un vector está siendo referenciado.
11.
```
30 FOR K = 1 TO 10
40 READ N (K)
50 NEXT K
```
12. `110 LET R (5) = 9.5`
13. Los subíndices pueden ser valores numéricos constantes, nombres de variables y/o expresiones.
14. No, los subíndices no pueden nunca ser números negativos.
15. Sí, los subíndices serán siempre números enteros.

16. Verdadero.

17.
```
210 FOR L = 1 TO 100
220 LET E (L) = L
230 NEXT L
```

18.
```
30 FOR E = 1 TO 10
40 READ T (E)
50 LET V (E) = T (E)
60 NEXT E
```

19.
```
70 FOR F = 1 TO 10
80 LET X (11 – F) = V (F)
90 NEXT F
```

20.
```
100 LET S = X (1)
110 FOR G = 2 TO 10
120 IF S <= X (G) THEN 140
130 LET S = X (G)
140 NEXT G
150 PRINT S
```

21.
```
160 LET L = X (1)
170 LET P = 1
180 FOR K = 2 TO 10
190 IF L >= X (K) THEN 220
200 LET L = X (K)
210 LET P = K
220 NEXT K
230 PRINT L, P
```

Capítulo 17

1. El método de la burbuja hace que el menor valor en un vector suba a la parte superior del vector justamente como una burbuja de aire en el agua sube a la superficie.
2. El primer paso está diseñado para trasladar el menor valor en el vector a la parte superior del vector.
3. Se requiere un máximo de n – 1 pasos donde n se refiere al número de valores en el vector a clasificar.
4. La terminación definitiva se señala cuando algún paso no provoca un intercambio de valores consecutivos del vector.
5.
```
100 DATA 44, 24, 18, 56, 6, 78, 21, 27, 18, 34, 15
110 DIM A (11)
120 FOR K = 1 TO 11
130 READ A (K)
```

```
140 NEXT K
150 FOR L = 1 TO 10
160 LET S = 0
170 FOR M = 1 TO 11 – L
180 IF A (M) <= A (M + 1) THEN 230
190 LET T = A (M)
200 LET A (M) = A (M + 1)
210 LET A (M + 1) = T
220 LET S = 1
230 NEXT M
240 IF S = 0 THEN 260
250 NEXT L
260 FOR N = 1 TO 11
270 PRINT A (N)
280 NEXT N
290 END
```

6. Una tabla bidimensional es una tabla que puede ser pensada teniendo filas y columnas como dimensiones.

7. Hay 10 filas y 12 columnas en la tabla G.

8.
```
10 DIM H (4, 5)
20 LET C = 1
30 FOR J = 1 TO 4
40 FOR K = 1 TO 5
50 LET A (J, K) = C
60 LET C = C + 1
70 NETX K
80 NEXT J
```

9.
```
100 DIM L (4, 4)
110 DATA 1, 5, 9, 13, 2, 6, 10, 14, 3, 7, 11, 15, 4, 8, 12, 16
120 FOR N = 1 TO 4
130 FOR M = 1 TO 4
140 READ L (N, M)
150 NEXT M
160 NEXT N
```

10.
```
:
:
:      sentencias no señaladas
:
:
100 LET S = 0
110 FOR K = 1 TO 8
```

```
120 LET S = A (K, 2)
130 NEXT K
140 PRINT S
```

Capítulo 18

1. Una matriz es un cuadro de números. En BASIC esta disposición está normalmente en dos dimensiones.
2. La sentencia MAT READ sustituye cinco sentencias; dos FOR, un READ y dos NEXT.
3. 200 MAT PRINT W.
4. Las sentencias MAT READ y MAT PRINT requieren información de la dimensión cuando las dimensiones exigidas no son las señaladas en la sentencia DIM asociada.
5. Las dos sentencias obtienen información de la dimensión de las sentencias asociadas DIM.
6. 100 DATA 1, 5, 9, 13, 2, 6, 10, 14, 3, 7, 11, 15, 4, 8, 12, 6.
7. 110 MAT F = ZER.
8. Sí. La información de la dimensión en la sentencia MAT ZER puede ser redefinida.
9. La sentencia MAT deberá usarse con la sentencia MAT CON. Ejemplo:

   ```
   120 MAT G = CON
   ```

10. Sí. La información de dimensión en la sentencia MAT CON puede ser redefinida.
11. La sentencia MAT deberá usarse con la sentencia MAT IDN. Ejemplo:

    ```
    140 MAT H = IDN
    ```

12. Las cuatro sentencias MAT que permiten la redefinición de dimensiones son MAT ZER, MAT READ, MAT CON y MAT IDN.
13. 400 MAT T = R + S
14. 410 MAT W = U − V
15. 420 MAT X = L * M
16. La sentencia MAT MULTIPLY permite únicamente que dos matrices sean multiplicadas juntamente al mismo tiempo. L * M * N puede calcularse en dos pasos. (Ver la respuesta a la cuestión posterior 17).
17. 500 MAT Z = P *Q
    510 MAT T = Z * R
18. 520 MAT X = 6 * W

19. 500 MAT Y = INV (K)
20. 600 MAT D = TRN (C)
21. 110 MAT INPUT X
22. NUM toma el valor 5.

Capítulo 19

1. Las mismas reglas que se aplican a los nombres interpretados como tomando valores numéricos se aplican a los valores alfanuméricos, –excepto que un signo de dolar se añade al nombre alfanumérico.
2. Un valor alfanumérico puede constar de hasta 60 caracteres.
3. Sí. Los valores alfanuméricos pueden ser enteramente numéricos.
4. Sí. Los valores alfanuméricos pueden ser comparados.
5. La sentencia IF no señala relación ni presenta comillas alrededor de las palabras TEST CASE. Ella fue probablemente interpretada como

```
830 IF M$ = "TEST CASE" THEN 1500
```

6. Los valores alfanuméricos no pueden usarse en cálculos aritméticos, aún cuando los valores consten enteramente de dígitos numéricos.
7. Sí. Los valores alfanuméricos pueden ser dados en sentencias DATA.
8. Hay tres valores en la sentencia DATA: JAN, 25 y 1776.
9. Verdadero.
10. Verdadero.
11. Sí.
12. Sí.
13. El comando RESTORE$ permite que cualesquiera valores alfanuméricos en las sentencias DATA sean releídos.
14. Sí. Los nombres de los arrays alfanuméricos pueden ser una sola letra del alfabeto seguida por un signo de dolar.
15. El programa imprime:

```
ESTO ES UNA PRUEBA
UNA PRUEBA ESTO ES
```

16.

```
100 LET A$ = "ROGERS"
110 LET B$ = "WILLIAMS"
120 LET C$ = "BENSON"
130 IF A$ < B$ THEN 220
140 IF B$ < C$ THEN 170
```

```
150 PRINT C$, B$, A$
160 GO TO 290
170 IF A$ < C$ THEN 200
180 PRINT B$, C$, A$
190 GO TO 290
200 PRINT B$, A$, C$
210 GO TO 290
220 IF A$ LC$ THEN 250
230 PRINT C$, A$, B$
240 GO TO 290
250 IF B$ < C$ THEN 280
260 PRINT A$, C$, B$
270 GO TO 290
280 PRINT A$, B$, C$
290 END
```

17.
```
100 DIM G$ (10)
110 FOR K = 1 TO 10
120 READ G$ (K)
130 NEXT K
140 DATA KINDER, SCHULE, MEIN, DER, HIER
150 DATA KNABE, ALLES, ANFANG, IST, SCHWER
160 FOR K = 1 TO 9
170 LET S = 0
180 FOR N = 1 TO 10 – K
190 IF G$ (N) < = G$ (N + 1) THEN 240
200 LET T$ = G$ (N)
210 LET G$ (N) = G$ (N + 1)
220 LET G$ (N + 1) = T$
230 LET S = 1
240 NEXT N
250 IF S = 0 THEN 270
260 NEXT K
270 FOR P = 1 TO 10
280 PRINT G$ (P)
290 NEXT P
300 END
```

18.
```
100 PRINT "HOLA, ¿COMO SE LLAMA?"
110 INPUT A$
120 PRINT "¿CUAL ES SU EDAD? "; A$; "?"
130 INPUT G
140 PRINT "MUY INTERESANTE"; A$; "."
150 PRINT "¿CUAL ES SU ASIGNATURA FAVORITA?"
160 INPUT F$
```

```
170 PRINT A$;" CREO QUE PARA UN MUCHACHO DE"; G;
180 PRINT "AÑOS DE EDAD, EL TEMA DE "; F$
190 PRINT "ES MUY ADECUADO."
200 END
```

Capítulo 20

1. Una subrutina es un grupo de instrucciones que son semi-independientes y resuelven una parte de un problema largo. La subrutina puede llamarse para usarla cuando se necesita.
2. El comando GOSUB dice al ordenador que salte a una línea distante y que recuerde dónde está situado el comando de forma que el programa pueda ser transmitido atrás, precisamente a la sentencia siguiente al GOSUB.

   Un GO TO le dice al ordenador que salte incondicionalmente a una línea distante. El programa continúa a partir de dicha línea sin recordar dónde estaba situado el GO TO.
3. Una sentencia RETURN en una subrutina le dice la programa que vaya hacia atrás a la línea inmediatamente siguiente al GOSUB que llamó a la subrutina.
4. Los programas saltarán alrededor de las subrutinas; en otro caso el ordenador funcionaría en un comando RETURN y no conocería lo que tenía que hacer con él.
5. Sí. Un programa puede saltar a una subrutina desde distintos puntos del programa principal.
6. Sí. Una subrutina puede llamar a otra subrutina. (Las GOSUB de las subrutinas pueden estar anidadas.)
7. La sentencia ON proporciona una serie de saltos condicionales controlados por el valor asignado previamente a un índice variable.
8. X deberá tener los valores 1, 2, 3 ó 4.
9. El programa continúa con la siguiente sentencia en secuencia.
10. Sí. En un programa principal Vd. puede tener cualquier número de subrutinas que necesite.
11. Verdadero.
12. El programa cambia 2.3 por 2 y salta a la línea 100.
13. El programa cambia 2.8 por 2 y salta a la línea 100.
14. El programa va a la siguiente línea en secuencia.

Capítulo 21

1. La cinta de papel es útil como un medio de entrada porque puede usarse para leer un programa en la memoria más velozmente que la rapidez con que una persona puede escribirlo.

2. Guardar un programa sobre una cinta de papel es muy barato y cómodo; ya que almacenar un programa en el sistema de tiempo compartido cuesta $2.50 o más por mes.

Capítulo 22

1. Una función incorporada es un conjunto de instrucciones construidas en el sistema de tiempo compartido. Cada conjunto de instrucciones tiene un nombre –tal como SQR (raíz cuadrada), SIN (seno), COS (coseno), etc.
2. Las funciones incorporadas son SIN, COS, TAN, ATN, EXP, ABS, LOG, SQR, RND e INT.
3. La función SIN da el seno trigonométrico de un ángulo en radianes.
4. Radianes.
5. El argumento de una función es el valor con que la función opera cuando calcula un valor.
6. Un argumento puede ser un valor numérico constante, un nombre variable, y/o una expresión.
7. La función COS da el coseno trigonométrico de un ángulo expresado en radianes.
8. La función TAN da la tangente trigonométrica de un ángulo en radianes.
9. La función ATN proporciona un ángulo medido en radianes correspondiente al valor de la tangente dado en el argumento.
10. El valor proporcionado a la función ATN es un valor numérico representando la tangente de algún ángulo.
11. 2.718281828...
12. La función EXP eleva el valor *e* al valor dado en el argumento.
13. Verdadero.
14. La función ABS proporciona el valor absoluto del valor dado en el argumento.
15. Verdadero.
16. La función LOG proporciona el logaritmo natural del valor dado en el argumento.
17. La función SQR da la raíz del valor dado en el argumento.
18. Verdadero.
19. Verdadero.
20. Verdadero.
21. La función RND da un número aleatorio. El número puede ser tan pequeño como cero y puede variar hasta un valor muy próximo a 1 sin ser efectivamente uno.

22. El comando RANDOM hace que el ordenador dé una serie no repetible de números aleatorios en un programa.
23. Verdadero.
24. La función INT da el mayor entero encontrado en el valor dado por el argumento.
25. El valor asignado a K es 3.
26. El valor asignado a K es 3.
27. El comando DEF se usa para definir una función para ser usada solamente en el programa en el que aparece la función definida.
28. Sí. Una función definida puede usarse exactamente en la misma forma en que se emplea una función incorporada.
29. Verdadero.
30. Verdadero.

Capítulo 23

1. Un fichero es un conjunto de valores numéricos que han sido conservados bajo algún nombre arbitrario.
2. Un fichero se construye escribiendo valores numéricos sobre una serie de líneas —cada línea teniendo su número— y a continuación guardando los valores.
3. Los dos modos operativos para un fichero son READ y WRITE.
4. El modo operativo inicial de un fichero es READ.
5. El comando es SCRATCH. Ejemplo:

```
250 SCRATCH #3
```

6. La sentencia FILES da los nombres de los ficheros que se usarán en el programa, y ella dice qué fichero es #1, cual es el #2, etc.
7. La sentencia es READ. Ejemplo:

```
300 READ #1, A, B, C
```

8. La sentencia es WRITE. Ejemplo:

```
350 WRITE #3, X; Y; Z
```

9. Se puede consultar a un fichero para "end of file" (fin de fichero) usando el comando IF. Ejemplo:

```
380 IF END #2 THEN 900
```

10. Sí. Ejemplo:

```
400 WRITE #3, P; Q; R; S
410 PRINT P, Q, R, S
```

11. El comando RESTORE cambia el modo de un fichero de WRITE a READ. Ejemplo:

```
500 RESTORE #3
```

12. Se puede examinar una salida de fichero llevándolo al espacio de trabajo usando el comando del sistema OLD y a continuación listarlo usando el comando del sistema LIST.

Indice alfabético

B A S I C

Curso acelerado

Este texto sobre el lenguaje de programación BASIC, escrito de forma asequible y clara, está dirigido a aquellas personas que deseen aprender rápidamente este lenguaje. Supone que el lector no conoce nada sobre la materia; por consiguiente, avanza desde los conceptos más fundamentales hasta los más sofisticados, gracias a lo práctico de su exposición de contenido.

Más de 350 ejemplos y ejercicios totalmente desarrollados y resueltos refuerzan los conocimientos teóricos, y permiten una verdadera autoformación.

EDITORIAL PARANINFO S.A.

Magallanes, 25 - 28015 Madrid

ISBN: 84-283-1289-3